NPS52-80-003

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

PETRI-NET SIMULATIONS
OF
COMMUNICATIONS NETWORKS

by

Stephen Craig Jennings

Robert John Hartel

March 1980

Thesis Advisor: Lyle A. Cox

Approved for public release; distribution unlimited

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral J. J. Ekelund
Superintendent

Jack R. Borsting
Provost

Reproduction of all or part of this report is authorized.

Released as a
Technical Report by:

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-80-003	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Petri-Net Simulations of Communications Networks		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1980
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Stephen Craig Jennings Robert John Hartel		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93940		12. REPORT DATE March 1980
		13. NUMBER OF PAGES 178
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Simulation
Petri-Net
Evaluation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Current technologies in the fields of telecommunications and computer processing are becoming increasingly integrated to the extent that "distributed" computer networks are assuming key roles in communications. The complex computerized systems necessary to support modern military command and control requirements are expensive. Designing such systems by trial and error is not feasible, yet no other viable alternatives exist. This thesis offers an original methodology for evaluating the predicted

DD FORM 1473, 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
(Page 1) S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

performance of military automated systems. Using Petri-Nets as a modeling tool, computer simulations with color graphics output are performed to demonstrate the feasibility of this approach as a systems design tool.

PETRI-NET SIMULATIONS OF COMMUNICATIONS NETWORKS

by
Stephen C. Jennings
Captain, United States Marine Corps
B.S., United States Naval Academy, 1971
and
Robert J. Hartel
Captain, United States Army
B.S., Texas A&M University, 1972

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY (C3)
(Stephen C. Jennings)
MASTER OF SCIENCE IN COMPUTER SCIENCE
(Robert J. Hartel)

from the
NAVAL POSTGRADUATE SCHOOL
March, 1980

ABSTRACT

Current technologies in the fields of telecommunications and computer processing are becoming increasingly integrated to the extent that "distributed" computer networks are assuming key roles in communications. The complex computerized systems necessary to support modern military command and control requirements are expensive. Designing such systems by trial and error is not feasible, yet no other viable alternatives exist. This thesis offers an original methodology for evaluating the predicted performance of military automated systems. Using Petri-Nets as a modeling tool, computer simulations with color graphics output are performed to demonstrate the feasibilty of this approach as a systems design tool.

TABLE OF CONTENTS

I. INTRODUCTION

As military planners look forward to the design of future command, control, and communications (C^3) systems, several key factors should be at the forefront of their thinking. First of all, it is readily apparent that there has been a proliferation of computer resources for military applications. Even the most "tactical" of systems today is becoming a sizeable collection of computers, databases, sensors and information-handling equipment.

Secondly, one can sense that the fields of telecommunications and computer science are becoming increasingly integrated. Although these once-separate disciplines have very different histories and traditions, they are experiencing a technological convergence which is having far-reaching implications for both the military and civilian societies. Today the concepts and techniques of computer processing have been integrated with communications to the extent that both fields share the same kind of logic, storage, switching and transmission. Because information handling systems now employ telecommunications and information in such an intimate mixture, it is difficult to distinguish what in the system is computer processing and what is communications.

A third key factor should be a realization that certain concepts of computer communication networks and distributed

system architectures offer advantages for military applications due to the potential for survivability and lack of centralization.

Computer networks are often created spontaneously by combining computers and communications. The growth of computer networks is one of the significant outcomes of the convergence of the two disciplines. An extensive array of computing resources can be connected over a wide geographic area via telecommunications channels. The potential architectures for such networks are limitless when one considers the variety of hardware, software, protocols and geographic distributions that might comprise the system.

Another key factor is that system planners need to focus their efforts upon total system integration. Tactical automated systems tend to be developed individually to meet unique mission requirements. For instance, separate systems are typically justified and developed for missions of fire control, air defense, intelligence, personnel management and logistics, etc. While these separate systems may peform satisfactorily alone, there is difficulty in providing a suitable management information system by which the overall commander or decision-maker can have access to the information in all these separate systems in a format that is easy to understand and use.

Another important consideration is that the proliferation of automated systems places a severe burden upon the communications equipment that links systems

together. There is a tendency for automated systems to be separately developed with insufficient emphasis placed upon the communications equipment that will transmit the information. In other words, the sensors and processors of a system may work splendidly, but planners must not take it for granted that the data will arrive at the correct destination in an error-free condition.

The communications equipment and channelization that carry the information must be as carefully engineered as the other components of the system. In addition, the data on the channel must be in a format that is compatible wth other systems. The U.S. Army is coming to grips with the fact that if the some 50 tactical automated systems on the drawing boards were to be fielded for use at the corps level, there exists no communications equipment capable of carrying the vast volume of information these systems would generate. In addition, when the communications equipment must operate with specifications of transmission security, jam resistance, and low probability of intercept in a severe electronic warfare threat environment, the design difficulties are considerably magnified.

Lastly, military planners should be concerned about being able to accurately predict system performance. An accurate predictor of system performance is needed for use by those in procurement duties to ensure that performance specifications given by contractors will in fact prove to be true when a new system is fielded. Managers of currently

operational systems also require this service. They are asking in the course of daily duties such questions as: What would be the effect of increased buffer space at this busy location? What would be the impact on total system performance if a particular node or link in the network is removed? Such a desire for prediction of system performance has created great emphasis upon modeling techniques and computer simulations of systems to answer these questions.

All of the above factors have given impetus to this thesis. A particular modeling tool which addresses these needs is described (the Petri-Net) and is implemented to facilitate communications network modeling.

The paper presents three primary points of original work:

1. It is demonstrated that automated networks can be meaningfully modeled with the use of Petri-Nets.

2. It is shown that Petri-Net models of networks can be adapted for effective computer execution and display on a color graphics terminal. Simulations which incorporate graphics output are more easily understood and have considerable educational value.

3. The results of this research indicate that the implementation of such a modeling technique in a production environment as a predictor of system performance is feasible.

The above 3 points, although successfully implemented in the Naval Postgraduate School C3 laboratory, represent a

new area of research in a preliminary stage of development. Future investigation is required to expand and validate this approach.

II. NETWORKS: MOTIVATION, TAXONOMY AND PERFORMANCE

A. INTRODUCTION

Predicting the performance of an automated network can indicate a measure of a system's effectiveness and efficiency.[3] Evaluating a system's performance is a complex task. This task often requires modeling. Many performance modifications are more suitably performed on models than the actual system, because "trial and error" production is not feasible economically. This chapter discusses the motivations for network design and the practicalities of predicting network performance.

B. MOTIVATION FOR THE ANALYSIS OF PERFORMANCE

The magnitude of information processing in the United States is unprecedented and still growing. Computer processing and communications make a major portion of currently accessable information available to federal agencies and commerical businesses. As the country is becoming increasingly dependent on the need for information, the existance of reliable, effective computer communication networks is essential to transport computer-processed information.

With the added cost of energy, the attractiveness of moving resources to the user via computer communications networks is apparent.[5] The performance evaluation of these networks is, therefore, a measure of the system's ability to

meet user requirements.

Adequate performance evaluation tools currently do not exist. Therefore, decision-makers responsible for selecting new systems usually rely on the designer's claims as to performance, and procure systems accordingly.

Both the civilian and military communities place very heavy demands on communication's facilities during crisis situations. Throughout history existing systems have often not been sufficient to carry such communications traffic. [8]

The civilian community shares these problems. Many networks are engineered to carry mean traffic loads and are not planned for crisis contigencies. The commerical telephone network, for instance, is inundated with traffic Christmas and Mother's Day, severely degrading system performance. The disruptive affects of a national emergency on all types of automated networks can only be imagined.

In the Defense Communications Agency (DCA), work is presently under way to develop tools for evaluating the performance of planned computer communication networks. This effort is underway because development of these expensive systems by "trial and error" can no longer be afforded. The necessity to have confidence in the system's ability to meet design specifications before production is essential, since normal federal procurement cycles stretch out over eight to ten years. This confidence can be insured by utilizing effective performance evaluation tools.

C. TAXONOMY

A great deal of ambiguity exists in the jargon of the networking field. [7] Therefore, several recurring network descriptions are defined in this chapter to provide a consistent vocabulary. Using these definitions, network issues can then be succinctly conveyed to the reader.

1. Networks

The term "network" conveys the concept of individualized cells and a degree of interconnectivity. A network exists for the purpose of achieving a desired objective. The individual characteristics of networks are related to network topology, hardware configurations, and software control features designed to accommodate the user's requirements.

2. Computer Communication Networks

A "computer communication network" is a system consisting of one or more computers and terminals, and a communications subsystem which connects them. [7] The primary purpose of this network is to facilitate the efficient flow of data, and provide the required supportive processing functions. The communications subsystem consists of transmission facilities and associated communications processors. Communications processors are computers dedicated to exclusively handling communications tasks.

The classification of computer communication networks often centers around the network topology, network connectivity, switching protocol and the degree of

implementation of system-wide control features.

The term "centralized computer communication network" is used to define a network that possesses a high degree of centralized functions. Another network classification, "distributed systems", is used to describe a low degree of centralized functions.

The differences between centralized computer communication networks and distributed systems reveal themselves in the degree to which system functions are distributed. There are no concise metrics which delineate the exact classification of a network. The interpretation is subjective.

Further elaboration on specific advantages and disadvantages of the distribution of system functions and the definition of distributable functions can be found in CYSPER [6].

Centralized computer networks essentially formalize the system control structure into pre-selected system control nodes. These nodes contain what is often referred to as the network control programs (NCP). The NCP can control the processing, database management and communications management functions. This type of networking characteristically has a low degree of fault tolerance because network direction eminates from only one node imposing network restrictions should this node be damaged.

Distributed systems, however, are typified by the distribution of system management functions. Although no

"pure" distributed systems exist in reality, the term is widely used to indicate a high degree of distributed functions within a network. This "peer" structure acts in a cooperative sense. Routing algorithms, for example, rely on information that is "cooperatively" passed from node to neighboring node, thereby deriving information, not on a global basis but on a localized one. Distributed systems have a high degree of survivability and are, therefore, more fault tolerant than centralized computer communication networks. The advantage of distributed systems is, however, partially offset by the increased "overhead" necessary to coordinate system functions.

3. Further Reading

Numerous terms used in networking contribute to confusion due to the lack of an industry-wide standardized taxonomy. The intent here was to define certain keywords used throughout this thesis. Further clarification of system's taxonomy, although important, is not discussed. Taxonomical studies recommended are [2], [9] and [23].

D. PERFORMANCE PREDICTION AND MODELING

"Performance" is defined as the degree to which the system fulfills user requirements. In terms of networks, these measures are often determined by the network's workload capacity or throughput in the sense that the network can first perform the desired functions and secondly perform with a degree of timeliness. [4]

The ideal means of measuring network peformance is to extract data from the system itself. The collection of data from existing networks is often difficult. The performance testing of networks under heavy traffic loads or those operating in a degraded mode is a sensitive matter, because interference of the monitoring equipment can not be tolerated. Networks in design phases, of course, can not be measured and require alternative assessment methods.

The modeling of existing and planned networks has become an important component of performance evaluation. As an added benefit, modeling also facilitates the understanding of a system's design and interrelationships.

The modeling process can follow one of several different techniques or a combination thereof. The primary techniques are: 1) mathematical modeling and 2) simulation.

Mathematical modeling employs theories of queuing and flow by describing certain network characteristics in sets of equations. The process is, however, complex and often assumes away critical parameters. The primary disadvantage with mathematical modeling appears to be just this problem of assumptions. Too many assumptions impose an unacceptable degree of abstraction. Although the validity of mathematical modeling techniques has been confirmed [4]. the methodology is often understandable only by the modeler.

Simulation, the second alternative, leads the modeler to numerous techniques. A simulation is an abstraction of concepts pertinent to the problem being studied, and upon

which the modeler can apply varying experimental variables. The model simulates only those features the modeler feels are relevent to the problem. Herein lies the critical danger of simulation. The danger may best be expressed by the question, Does the model bear relevancy to the real problem? [10]

The major problem of simulation, as well as analytical models, is therefore the validation of the model. Many simulation experts talk in terms of performance reliability factors but fail to state that these factors may be just derived from outputs of a model, and the closeness of the model to reality may or may not be substantiated.

The modeling process itself consists of the construction of the model, followed by the validation of the model, and finally modifications to the model based on results of the validation process.

Once the model has been defined, simulations are run with the model to evaluate network behavior. The problem is to predict peformance of real networks by evaluating behavior on the network models. The simulations then provide a means by which network design deficiencies can be identified and corrected.

Another issue in simulation analysis is the area of overdesign. Simulations should indicate those systems components that do not add to the capabilities of the network. This analysis can measure the device utilization of specific components within the network.

The key measurement of a computer communication network is the network's workload capacity or throughput. Throughput is generally measured in number of message units per time period, and provides a measure of effectiveness and efficiency of the system. The parameters involved in throughput are: 1) network configuration (topology), 2) network control algorithms, and 3) network reliability.

E. SUMMARY

A solution to predicting performance in systems where the collection of data is difficult or the system is in design stages, is to build a model of the system. The model could then be tested over the entire sprectrum of performance specifications.

The goal of performance evaluation is the prediction of the degree to which the system fulfills the intended objectives. The major concern in computer communication networks is the degree to which the network can perform the task and the degree of efficiency with which the task is completed. A by-product of the evaluation should be the identification of areas of over and under-design. Once design failures have been identified, design tradeoff decisions can then be made. This process of performance analysis is aimed at optimizing the existing or planned network's performance and ensuring that the performance meets the contracted user requirements.

III. MILITARY APPLICATIONS OF DISTRIBUTED SYSTEM TECHNOLOGY

A. INTRODUCTION

This chapter discusses current military programs and research efforts that are applying the concept of automated networks to tactical missions. It is written to give the reader some background information on programs and terminology. With this background, the applicability of the simulations described in later chapters will be clearer.

B. PACKET SWITCHING

The transmission of computer to computer digital messages has had significant impact on communications switching techniques. In fact, the concept of packet switching was invented to a large extent because of the unique requirements of computer based systems. Packet switching was designed as an alternative to circuit switching.

The circuit switching technique of older communications systems is a method of establishing a route for communications traffic whereby a complete link between the calling and receiving station is set up and maintained exclusively for the exchange of those two stations. The connection is maintained until one of the stations breaks off transmission or reception. A technique such as this tends to be wasteful in computer communications because computer communications are typically "bursty" in nature;

that is, the messages are very short in duration and require fast responses.

Packet switching is designed to make efficient use of a communications channel when the traffic is bursty. [11] In this technique messages are divided into discrete "packets." A packet is a block of information containing a fixed number of bits. Each packet contains the text of the message plus a control header. The header contains enough information (for example, source, destination, routing plan, message sequence number, etc.) to guarantee the packet will arrive at the proper destination. In addition, there will usually be some checks on each such block, so that any switch through which the packet passes may exercise some degree of error control.

Figure 1. shows a typical composition of a packet of information bits. This particular example is taken from a packet radio network. [16]

In a packet-switching network, the packet represents the fundamental unit of transportation. One message may be broken into several packets and each packet may be independently routed to its final destination. Of course, at the destination the packets must be re-assembled in the correct order to reconstruct the original message.

Because packets of the same message can be sent by different routes, congestion on the network can be decreased. Each packet contains its own control information in the header, and there are no lengthy connection and disconnect times as in the case of circuit switched systems.

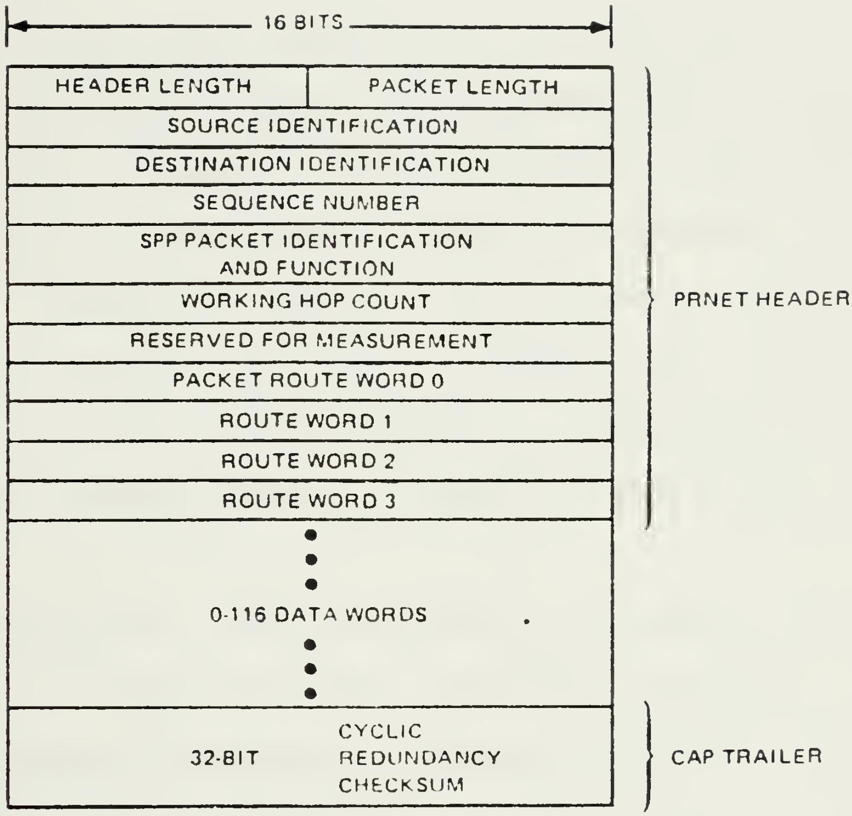

FIGURE 1 FORMAT OF PRNET PACKET (PRCAP3)

It should be noted that messages which are broken into packets are only meaningful within the network. When packets are passed through gateways into other packet-switched networks, a new intra-net level of protocol is required.

C. THE ARPANET

Perhaps the best known example in the military of a large scale distributed system is the ARPANET. This research effort has been sponsored by the Defense Department's Advanced Research Projects Agency (ARPA). The ARPANET is a non-secure, packet-switched, distributed computer communication network which links together the computing facilities at universities and military installations across the continental United States and reaches overseas to London and Hawaii.

The justification and advantages of a network such as the ARPANET are summarized in the following excerpts from the text <u>Computer-Communication Networks</u> written in 1973: [1]

"One of the most successful aspects of the experiments in the use of time-shared computer systems conducted during the past decade was the ability to share computing resources among all the users of the system. Controlled sharing of data and software, as well as the sharing of the time-sharing system hardware, has led to much higher programming productivity and better overall utilization of the computing and user resources."

Even in these time-sharing systems, however, the system capacity was simply not large enough to perform all the storage requirements and computing potential that a decision maker required. There was the lack of a large enough community (critical mass phenomenon) in a single application area. Although it is possible to physically transfer programs or data from one community to another, this causes restrictions in language standards and hardware systems.

To quote further: "A viable alternative to program transferability, while permitting full resource sharing, is to provide a communications system that will permit users to access remote programs or data as if they were local users to that system. In addition, it should be possible for a user to create a program on his local machine that could make use of existing programs in the network as if they were available on his local machine. Rather than trying to move the programs from machine to machine, the network would allow the user or his program to communicate with a machine on which the program already executes. If enough machines can be connected into such a network, the total community in any particular application area would be sufficiently large enough to reach critical mass." [1]

This, essentially, is the rationale for military applications of the ARPANET research, both on the strategic and tactical level.

D. MILITARY APPLICATIONS

We have spoken in a previous chapter of the tremendous growth of automated data systems. Tactical data systems, be they for the purpose of intelligence, pesonnel management, fire direction, logistics, or command and control, tend to be engineered and developed separately without the consideration of total military mission. Nevertheless, the battlefield commander cannot make wise decisions based on input from just one of these systems. There is a good deal of interplay between all of these functional areas, and all must be considered.

One single tactical computer in a single command post could not be built to store and update all of the data represented in the combined systems. And if it could, such centralization would be unwise. The answer seems to be a distributed computer network built to interconnect and share the resources of the individual systems. This means overlaying an ARPANET-like architecture onto a series of distributed processors on the battlefield.

The ARPANET distributed architecture geographically separates data bases and computing resources. This distribution tends to decentralize a network, moving from a traditional hierarchical configuration to a grid or mesh-type configuration. Such decentralization is important to overall system survivability and reliability. A network architecture such as this is capable of remaining operational if one or more nodes is rendered

non-operational. The architecture, combined with packet switching technology, allows for sufficient alternate routing capability to ensure a robust system. It offers significant improvement over some of the present "backbone" (hierarchical) systems in which the failure of one node along the chain would completely disrupt communications on the entire network.

It is obvious that a mobile and tactical application of ARPANET technology would offer new challenges to system engineering. Most apparent is the fact that ARPANET sites are interconnected by high speed, low-error, fixed telephone circuits. This kind of interconnectivity is not possible on a dynamic battlefield. The only other alternative is to utilize mobile, digital radio equipment to achieve connectivity.

E. THE ALOHA SYSTEM

Several years ago researchers at the University of Hawaii began work on such hardware. Because there was an unusually high error rate on the local telephone lines, remote users of the university computer were unable to effectively communicate with the computing facility. This led to a research program to investigate the use of burst radio transmission in place of telephone lines for error-free, line-of-sight communications to the computer center. The resulting effort became known as the Aloha System, a series of packet-switched, ultra-high frequency (UHF), radio terminals. [13]

The Aloha System is essentially a broadcast, multi-access network. The "broadcast capability" of a radio channel implies that a signal generated by a radio transmitter may be received over a wide area by any number of receivers. "Multi-access capability" of a radio channel means that any number of users may transmit over a common channel. Hence, all users within line-of-sight of one another form a network that is completely connected, independent of the number of users.

F. PACKET RADIO INTRODUCTION

Research done at the University of Hawaii led to the development of packet radio. Packet Radio extends the Aloha System to military uses. The goals of military experimental versions of a packet radio system differ from those of the original Aloha net in the following areas: [13]

(1) Distributed control of the network management functions should be provided among multiple stations for reliability, and the use of a netted array of possibly redundant repeaters for area coverage as well as for reliability should be included.

(2) The system should use spread-spectrum signaling for coexistance with other possibly different systems in the same band and for antijam protection. Surface acoustic wave technology has become a viable current choice for matched filtering in the receiver.

(3) The provision of autherticaion and anti-deception mechanisms is required.

(4) System protocols should be incorporated that perform network mapping to locate and label repeaters, route determination and resource allocation, remote debugging, and other distributed network functions.

(5) The use of various implementation techniques to provide efficient operational equipment such as repeater power shutdown except while processing packets should be included.

Because packet radios operate within a broadcast network, and all the network radios use a common frequency band (1710-1850 MHz), this technology has some favorable implications for frequency management and frequency conservation. According to current military doctrine, the frequency spectrum is allocated roughly in accordance with each user's stated requirement. In an Army or Marine Division this results in a frequency management problem of too many nets requiring too few frequencies and a constant threat of degraded communications due to mutual inteference problems. Once a frequency is allocated for a particular mission, it is not available for use by others in the same area.

This might be an effective management technique if each assigned band were actually used most of the time. In practice this is not usually the case, and much of the frequency spectrum is idle (not engaged in carrying traffic).

A broadcast network in which a number of users share a common broad frequency band offers improvement to this situation. The limited frequency spectrum could be used more efficiently if (1) the shared frequency band was wide enough to allow all users to transmit required traffic, (2) a channel access scheme was defined such that all users could access the channel when needed while at the same time allowing little or no mutual interference, and (3) the channel usage was high enough to ensure minimum empty time when the channel was not in use.

G. CHANNEL ACCESS SCHEMES

One primary means of categorizing radio broadcast systems is the method employed for channel access. As mentioned earlier, packet communications have found important applications in ground-based radio information distribution, and in this situation there exists a common broadcast channel that is available and shared by a multiplicity of users. Because these users demand access to the channel at unpredictable times, some access scheme must be introduced to coordinate their use of the channel in a way which prevents degradations and mutual interference.

A large number of channel access ideas have been invented, analyzed and described in current literature. For a summary of these schemes, see [14]. All of these schemes, however, might be placed in one of three broad categories. [15] Each category has its own advantages and costs.

1. Category I

The first category involves random access contention schemes whereby little or no control is exerted on the users in accessing the channel. This results in the occasional collision of packets on the air. A collision implies that at least one colliding packet is unintelligible and that channel usability for the time of the collision may be lost. Access schemes which fall into this first category are the pure Aloha, the slotted Aloha, and to a lesser extent, Carrier Sense Multiple Access. These are the access methods used by packet radio systems.

To better understand random access contention schemes, consider the example shown in Figure 2. There are some number of users, each of which transmits some number of packets of time duration "tau" at random times. Line four, labeled SUM, indicates the total traffic that all the users attempt to send in a fixed time.

In this example all traffic is able to be transmitted without conflict except for the collision indicated by the hashed area. These two packets (P-1 and K-1) may both be unrecognizable to the receiving station (at least one will be unreadable) and both could require retransmission.

In the non-slotted Aloha random access technique, packets are transmitted as soon as they reach the top of the transmit queue at the radio. No consideration is made of current channel activity. Therefore, they risk collision

with other packets on the air.

In the slotted Aloha method, time is broken into discrete quantities equal to the maximum time of propagation within the network. Users are restricted to transmission only at the beginning of each time slot. Again, collisions may be frequent.

In the carrier-sense mode, the radios listen before they talk and thereby reduce the risk of packet collisions. The radio senses the state of the channel before transmitting. If the channel is occupied, the radio waits a random amount of time and senses the state of the channel again before attempting to transmit.

An important measure of peformance for evaluating these random access techniques is "throughput." Throughput in packet radio technology is defined slightly differently than the definition given in Chapter II. Here it is defined as the percentage of time that the channel is actually occupied by useful traffic. Or, to put it another way, it is the message density on the channel. Is the channel carrying useful traffic (non-colliding packets) most of the time, or is there a lot of time wasted between packets when the channel is empty?

Figures 3 and 4 show how throughput is calculated in both techniques.

RANDOM ACCESS SCHEMES

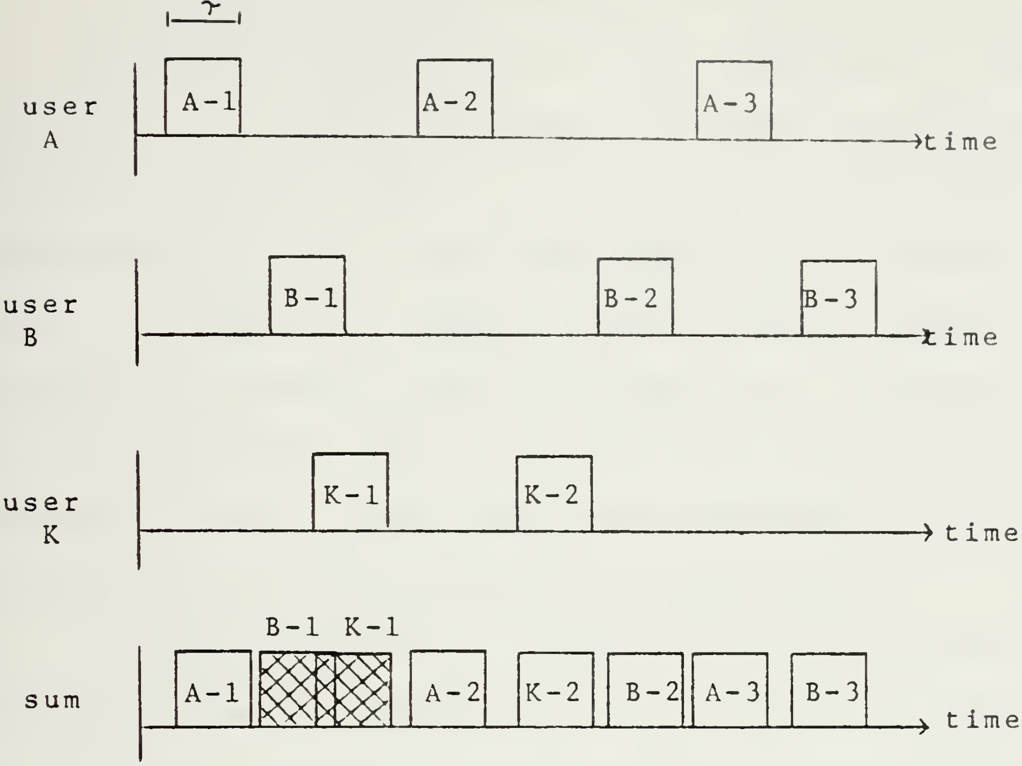

Figure 2

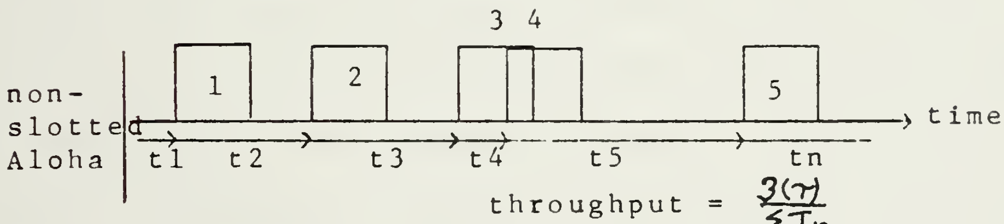

Figure 3

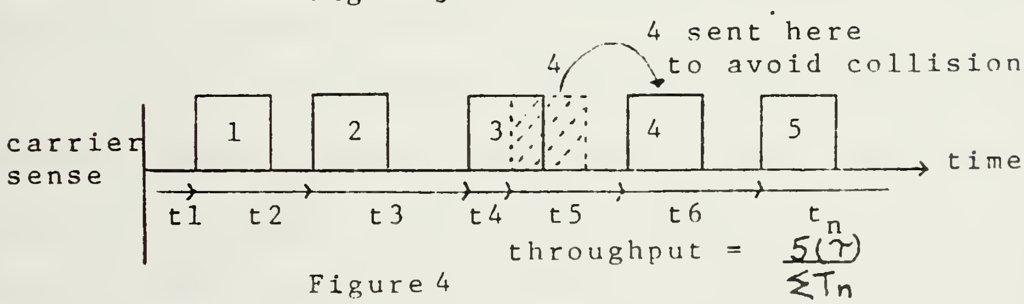

Figure 4

It should be obvious that throughput is higher in the carrier-sense mode due to the decreased number of collisions. In fact, analysis has shown that the maximum throughput possible with the non-slotted Aloha method is approximately 1/2e or 18.4%. Throughput in the slotted Aloha method is twice as high as the non-slotted method (36%). Throughputs as high as .80 to .90 have been obtained using the carrier-sense mode for cases in which the channel propagation time is small compared to message duration. Both of these methods are classified as asynchronous.

Another important measure of peformance for distributed computer systems is a low delay time. This means that queries and responses between the system and the user take a minimum amount of waiting time. A short delay time is important when there are many interactive users on the net. Typically, in order to decrease delay time packet length is shortened. Long packet lengths, however, are necessary to increase throughput, shorten queue lengths and decrease processing overhead per bit.

On the battlefield, calls for fire from a forward observer would typically be short messages requiring a fast response, whereas, intelligence summaries are normally several pages long and require no reply. If both types of messages are carried by the same communications channel, one can readily understand why there are tradeoffs between throughput and delay time.

2. Category II

At the opposite extreme of categories of radio channel access methods, there are schemes which use completely static reservation access methods. These schemes pre-assign capacity to users and effectively create "dedicated" as opposed to multi-access channels. Such schemes as Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA), and Code Division Multiple Access (CDMA) fall into this category. The Joint Tactical Information Distribution System (JTIDS), Phase I, and the U.S. Marine Corps Position Location Reporting System (PLRS) use the TDMA scheme in which time is broken down into discrete intervals. The largest time cycle (called an "epoch" in JTIDS) is divided into thousands of smaller time slots or windows. Each user on the broadcast network is assigned one or many time slots in which he can transmit messages. After the passage of the cyclic time period his time slot again appears and he can transmit again. This technique, obviously, is highly dependent upon all users maintaining accurate time synchronization. The FDMA and CDMA schemes have also been developed to statically assign each user a fixed portion of the channel according to a unique frequency or code respectively. Here the problem is that a bursty user will often not use his preassigned capacity, in which case it is wasted. When a user is idle his portion of the channel cannot be used by other stations with traffic.

3. Category III

Between these two extremes are the dynamic reservation systems which only assign channel capacity to a user when he has data to send. In these schemes a certain portion of the channel is set aside in which to dynamically schedule transmission times. Several schemes fall into this category. In a Polling scheme the user waits passively to be asked if he has data to send. In an active reservation scheme the user asks for capacity when he needs it. In the Mini-Slotted Alternating Priority scheme a token is passed among numbered users in a prearranged sequence, giving each permission to transmit when he receives a token. The cost of these schemes is the overhead required to implement the dynamic reservations.

The following Figure 5. summarizes the costs of these three categories. [15]

The Cost of Distributed Resources

Access Method	Collisions	Control Overhead	Idle Capacity
Random Access Contention	yes	no	no
Dynamic Reservation	no	yes	no
Fixed Allocation	no	no	yes

H. CURRENT PROGRAMS

As was previously mentioned, before large scale distributed data systems can be introduced to a mobile battlefield, higher capacity communications hardware is required. The U.S. Army is working on two programs to meet this requirement. The two systems under development are the PLRS/JTIDS hybrid and packet radio.

An Army Letter of Agreement [25] which addresses the need for these systems reads: "There is an urgent need for communications capable of supporting existing and programmed automated systems for Air Defense, Field Artillery, Intelligence, and Command and Control. Characteristics of machine-to-machine communications, coupled with the need for fast reaction times and a high degree of mobility, result in a requirement for a specialized distributed data communications system. Without this data communications improvement, highly sophisticated and highly effective weapon systems fielded in the early 1980's will not operate to full potential."

In further describing present communications systems this document speaks of existing equipment as being "technologically old, generally manually connected, too large and immobile (multichannel), and requiring intensive maintenance and logistical support." "The current communication system cannot, without the addition of a digital data communications capability, meet the demand imposed by the emergence of automated systems in the early

1980's."

The precise extent of this communications shortfall is a matter of intense study by the Integrated Tactical Communications Study (INTACS) Update Study effort.

The PLRS/JTIDS hybrid offers a solution to this problem in the mid-1980's. Packet radio, also, is a promising technology to satisfy future tactical data distribution missions. Its development schedule, however, effectively eliminates it as a short term candidate.

1. PLRS/JTIDS Hybrid

A detailed description of the PLRS/JTIDS hybrid is not possible here. Briefly stated, however, the system is a computer based system which provides real time, secure data communications, and position location and reporting information for tactical forces. It combines desirable features of both the PLRS an the JTIDS systems, using modified equipment from both systems. This system is planned for introduction to the field by 1986.

In addition to work being done on the PLRS/JTIDS hybrid, the Army is monitoring or participating in some interesting tests with packet radio, described in the following sections.

2. San Francisco Bay Experimental Packet Radio Network

The current experimental packet radio network being supervised by SRI International is located in the San Francisco Bay area and has been operational since July, 1976. Figure 6 [16] shows a map of the network sites.

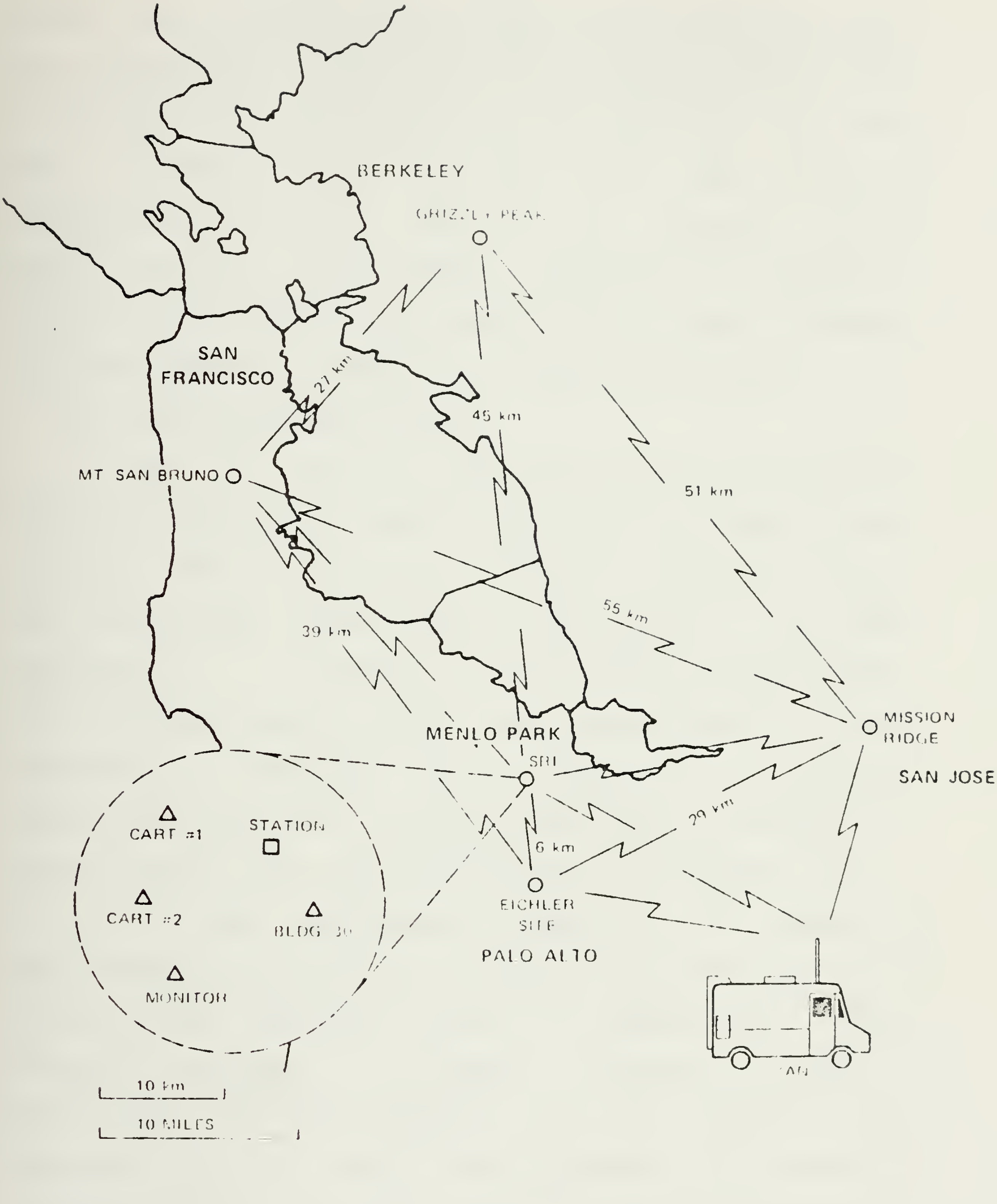

FIGURE 6 LOCATION OF MAJOR ELEMENTS OF THE PACKET RADIO TESTBED

There are two station PRU's located at the Menlo Park site. Each station has an associated PDP-11 computer attached. The network has four fixed repeater sites spread over the area and a variable number of packet radio user terminals (typically four to six). There are two vehicular packet radio terminals. These mobile terminals are an important aspect of the network, and a hand-off of a mobile terminal from one repeater to another is frequently exercised.

3. The Fort Bragg Test Bed

The U.S. Army has also recently set up a test bed for the evaluation of Packet Radio at Ft. Bragg, North Carolina. ARPA is sponsoring this effort, known as the Army Data Distribution System (ADDS). "The purpose of the ADDS experiment is to develop an environment in the resident XVIII Airborne Corps to permit user participation in the development, refinement and evaluation of innovative concepts for deployment of distributed data in support of future tactical Army data distribution requirements." [17]

In this multi-phased experiment, Ft. Bragg is experiencing a step-by-step build up of resources. The first phase of the experiment began in January, 1979. Three computer terminals, a network processor (called a TIP-Terminal Interface Processor) and a host computer were installed at Ft. Bragg and connected via commercial telephone lines into the ARPANET. After installation, operator training began in 1-2 day courses.

The second phase began in April, 1979 and the number of terminals was increased to fifty. The training in these phases acquainted operators of all ranks with the basic preprogrammed capabilities of the ARPANET including electronic mail, file management, directory maintenance, text editing, and printing. More specific applications geared to tactical information flow requirements are also being tested in garrison.

Phase III of the ADDS experiment is currently going on and involves the introduction of Packet Radio into the testbed. The PR network at Ft. Bragg will eventually grow to approximately 20 radios and 2 control stations. Initially, the radios are being employed in garrison to replace hard-wired connections. In the future, they will be deployed to the field in support of Corps exercises.

Packet Radio is expected to fullfill at least two major roles in the Ft. Bragg testbed. The first is to determine if this communications technology will satisfy the tactical data communications requirements of a corps on the battlefield. In this role, the system will be placed into the testbed as would any other communications system proposed for Army use. A second role is to provide a broad band communications channel for other systems under development which requires a data transfer capability. In this role the PRNET would provide communications for TACFIRE or some other intelligence or air defense system while maintaining its ARPANET connectivity as well. [17]

Reports from the Ft. Bragg testbed indicate that the XVIII Airborne Corps personnel have rapidly and enthusiastically adapted to the computer based communications technology. It is possible that the use of this data distribution technology can become as routine to commanders and staffs as voice communications are today. [17]

In any case, the Ft. Bragg experiment represents an innovative and unique approach to investigating advanced concepts in Army doctrine and tactics. The testbed is a departure from traditional Army tests designed to arrive at production decisions, and is being driven by the urgent need for increased tactical data distribution capability.

I. FUTURE IMPLICATIONS

Is is not an exageration to say that these are very crucial days for the U.S. military. Resource investment decisions are more important now than ever before in history, and the consequences of bad judgement offer potential for great loss. Some would argue that an ever-increasing dependency upon technology by the armed forces is a very dangerous trend. Nevertheless, it is apparent that electronics and telecommunications advances are beginning to have significant impact upon strategy. Chapter One of the U.S. Army Combat Communications Field Manual is entitled "Command, Control, and Communications (C3)". A quotation from this document helps to stress the intensity of future battles and the necessity of accurate

and realistic planning for that engagement. [24]

"The U.S. Army has arrived at a point where technology and reality have outrun our old tactics on fighting and left them in the dust. We have come to the shocking realization that the old way of doing things will not work any more."

"A good example of the change in combat reality facing today's soldier is an often used statistic from the Arab-Israeli War of 1973. In 20 days, over 1700 tanks were destroyed between the two sides. That's as many tanks as there are in five U.S. armored divisions. Technology has improved the weapons systems to the point where a tank has a 50-50 chance of being hit by the first round fired at it. We must retool our tactics to meet the reality of the next fight."

Certainly the capabilities of an army's command and control system has a great influence upon the capability of the force as a whole. The trends and technolcgies described in this chapter have far-reaching implications to doctrine and pose some problems that have yet to be solved. It is beyond the scope of this thesis to dwell on these implications in great length. A listing of the most obvious ones, however, is interesting and instructive.

1. <u>Chain of Command</u>

First of all, how will distributed systems change or affect the traditional chain of command structure? C3 system architectures typically reflect the chain of command within an organization, and this results in most C3 systems being

very hierarchical in structure. Distributed systems tend to be grid-like, peer structures. In a distributed system, is it advantageous to practice the concept of "skip echelon" reporting, in which certain levels of command may send traffic directly to the highest or lowest elements without intermediate "information only" stops? Certainly this would tend to decrease the time of delivery of messages and would avoid congestion on the network. But can the intermediate commanders afford to miss information that might prove to be critical or essential? What affect do distributed architectures have on the traditional role of communications centers and message centers? Would the requirement for these functions be eliminated? When the network is organized in a peer structure, how are protocols designed in order to preserve a "priority" system to message traffic? These are questions that remain to be answered.

2. Network Management

Next, a very important question is the manner in which network control and data base management is exercised. Although distributed systems appear outwardly to be decentralized, a static, inflexible network control design would make the so-called "distributed" system as vulnerable as its hierarchical predecessor. Clearly the responsibility for network management and data base updating must be transferable within the network, and the capability should be shared by more than one station. There is a tradeoff reached, however, between one station and multi-station

operation. The more nodes there are which can exercise network control, the more vulnerable the entire network becomes to spoofing and deception techniques, and the more costly and complex it becomes. It is interesting to note also, that some of the hardest decisions and longest delays in the JTIDS program have dealt with this question of network management.

3. How Much Redundancy?

In addition to these important questions one must ask also, "How much redundancy is enough?" Although distributed networks are more survivable and redundant, what kind of back-up systems are still required? Certainly an increased dependency upon satellite communications in today's world has decreased the investment in long-haul, high frequency (HF) communications systems. Some have argued with convincing reasoning that this is a dangerous position. It is generally more attractive to engineers and planners to invent and employ new systems rather than to improve the old. Nevertheless, it is necessary to retain and maintain older, proven hardware as back-up, secondary equipment. The question remains, "How much is enough?"

4. Propagation Loss Due to Higher Frequency

Because of the congestion existing at lower frequencies and the high bandwidth requirements of new automated systems, new communications equipment is being designed to operate at very high carrier frequencies. This restricts propagation to strictly line-of-sight distances.

While this situation is acceptable in ground-to-air and ground-to-satellite communications, it poses serious constraints upon ground-to-ground mobile communications, especially in rugged or forested terrain. The amount of propagation loss in this situation is highly significant, and the utility of these systems has yet to be proven.

5. Management Information Systems

It was mentioned previously that the development of an effective Management Information System (MIS) to integrate and display data to a commander and his staff is a formidable task. The Army, in fact, has been wrestling with this problem for over twenty years while attempting to field its Tactical Operations System (TOS, which is, in essence a MIS). After that amount of time, one questions whether such a system is nearer completion now than it was in 1960.

6. Interoperability

The military is fast finding out that interoperability means much more than mutually compatible equipment. It also means compatability of procedures, software, and message formats. One should watch closely the continued development of JTIDS and attempt to judge the success of joint service programs. The interoperability requirement tends to introduce system complexity in an attempt to make the system "all things to all people." When one moves to the problem of interoperability in the NATO environment, the question again becomes, "How interoperable can equipment be without becoming too costly, too bulky and

generally ineffective?"

7. Voice vs. Data Circuits

Another important question yet to be resolved is the proper trade-off of voice and data circuits. Again, JTIDS can be the case in point. Some services seem to be side-stepping procurement committments partly due to a lack of definition of system capability in this area. Although tactical commanders tend to prefer voice channels, voice channels require an enormously greater bandwidth allocation than data channels. If a commander is given the choice of having one voice channel or ten data channels into his command post, which will he choose? Which should he choose? What will the system offer?

8. System Cost

Finally, the question that will be asked most often is simply: "How much will the system cost?" Costs are divided into at least four categories. There are hardware and software costs, (it is common knowledge that the latter are now a greater consideration than the former) and there are initial procurement costs and life cycle costs.

Included in these costs is the manpower question. Is it realistic to think that as systems become more and more highly technical that the personnel who fix, operate and manage the systems will be better educated and more professionally competent?

IV. AN INTRODUCTION TO PETRI-NETS

A. INTRODUCTION

The purpose of this chapter is to introduce the reader to the particular modeling tool which forms the basis for this research. The history of Petri-Nets is first discussed. Then an explanation of how Petri-Nets work is presented. Following some simple examples, the chapter concludes with a brief summary of the strengths and weaknesses of this modeling tool.

B. HISTORY

The Petri-Net is named after its discoverer, Carl Adam Petri. These nets were developed in his early work in 1962 in Germany. They soon came to the attention of Anatol Holt who was then leading an Information Systems Theory Project for Applied Data Research, Inc. The work of this group eventually led to the theory of "systemics" [5] which dealt with the representation and subsequent analysis of systems and their behavior. At this point the modern formalism and notation of Petri-Nets was introduced. Holt also demonstrated the usefulness of the Petri-Net model in the representation of systems characterized by concurrent processes.

Perhaps the single largest source of research and literature regarding Petri-Nets has been Project MAC at the Massachusetts Institute of Technology. The Petri-Net model

was introduced to the researchers at Project MAC due to the association of Holt's group to J. Dennis' Computation Structures Group. This group has produced several PH.D thesis, together with many reports and technical memos dealing with Petri-Nets. In addition, MIT has sponsored two important conferences dealing with Petri-Nets. The first was the Project MAC Conference on Concurrent Systems and Parallel Computation held at Woods Hole in 1970. The second was the Conference on Petri-Nets and Related Methods, held at MIT in 1975.

This work, begun at MIT and continuing at other centers in the United States, until recently tended to concentrate on the formal or mathematical aspects of Petri-Nets. This work bears resemblance to the research in automata theory. It attempts to analyze systems by representing them as Petri-Nets, formally manipulating the representation in such a way as to derive information relating to the behavior of the modeled system. Because of the simplicity and power of Petri-Nets, they are excellent tools to use in the analysis of concurrent or asynchronous systems. They are finding their way into a number of diverse applications.

Petri, himself, is still actively researching, expanding his original theory. His extensions have led to a form of general systems theory called "net theory". Holt is continuing his research, concentrating on system representation and analysis of the formal representations. [5]

C. HOW PETRI-NETS WORK

Simply stated, a Petri-Net is a model. More specifically, it is an abstract, formal model that analyzes the flow of information in systems. [5,19] Petri-Nets also describe not only the information flow, but the controls and constraints of such flow. A Petri-Net graph models the static structure of a system in much the same manner as a flowchart models the structure of a computer program. As a modeling tool, Petri-Nets are especially useful in modeling systems that exhibit asynchronous and concurrent activity.

A Petri-Net consists of a collection of "events" and "conditions." In graphical notation, conditions are conventionally represented by circles and events are represented by bars. The Petri-Net is given structure and the capacity for interaction by connecting events and conditions with arrows.

An arrow from a condition to an event signifies an input condition to that event and implies that every occurence of the event terminates the "holding" of that condition. An arrow from an event to a condition signifies an output condition, and in this case, the occurence of the event commences the holding of the output condition. The graphic notation for a condition which holds is the marking of that condition by a "token".

The behavior of a system may be thought of as the occurence of events as time progresses. If all input conditions to an event hold, the event can occur. This

results in the holding of the event's output conditions.

Conditions may also be called "places" or "nodes" and events may also be referred to as "transitions." In the Petri-Net model of a system, directed arcs connect places to transitions and transitions to places.

The Petri-Net is first given a particular structure of places and transitions, and then it is "executed." Execution is governed by a "firing" protocol.

Simply stated, a transition may "fire" (symbolizing the occurence of an event) when all input conditions or places into that transition are marked with a token. When all inputs into a transition are marked, the transition is said to be "enabled". Figure 7 shows an "enabled" transition.

Execution of the Petri-Net involves the cyclic checking of all transitions once during each time interval. Each transition that is found to be enabled is fired, and tokens are moved from the input places of the enabled transition to the output places of that transition. This procedure continues for a set number of iterations. The flow of tokens in the Petri-Net thus symbolizes the flow of information or control in the modeled system.

By devising special methods for marking the number of tokens at the Petri-Nets nodes, the system status can be accurately and effectively recorded. The state of the system is reflected by an ordered set of mark status indicators which correspond to the nodes of the graph structure.

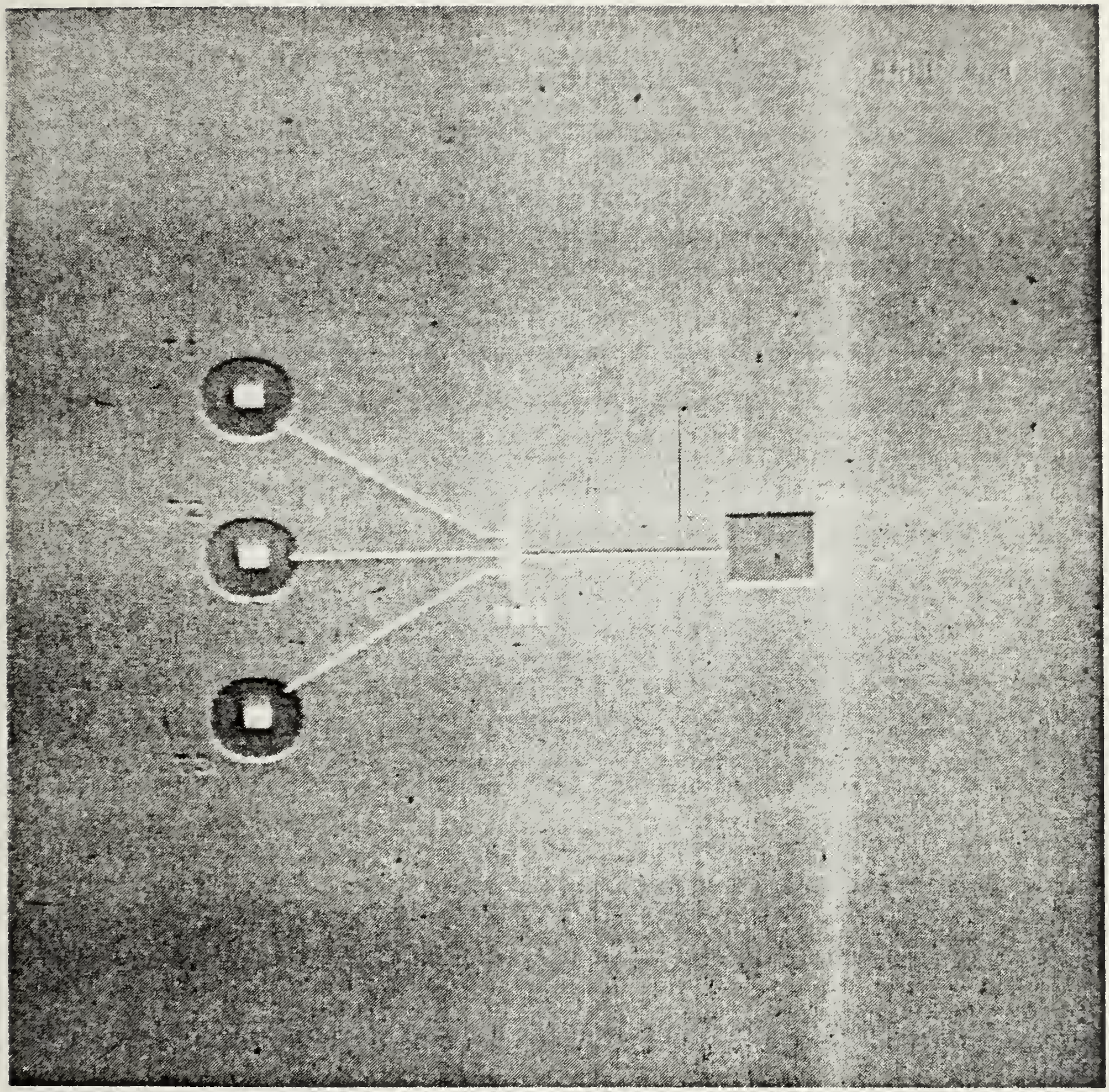

Figure 7.

In this thesis such an effective marking method is uniquely employed to give the viewer an accurate snapshot of network status.

Petri-Nets have rapidly gained acceptance over the last decade. Along with this aceptance has been the furthering of the understanding of Petri-Net properties.

D. A SIMPLE EXAMPLE

Figures 8 through 11 show the various states of a simple Petri-Net as execution occurs during four successive time intervals. [21] Notice four places (P1, P2, P3, and P4) and three transitions (TR1, TR2, and TR3). The directed arcs denote the interaction and relationships between input and output conditions. For instance, TR1 will become enabled when P1 (its only input condition) is marked with a token. At the time that TR1 fires, the token will be removed from its input condition (P1) and placed in its two output conditions (P2 and P4). In this manner flow of information or control is followed through the modeled system. Figure 8 shows the network at time = Ø with one token placed in P1. Figures 9 through 11 depict the Petri-Net as it continues execution through time = 3.

E. ADVANTAGES AND DISADVANTAGES

The following characteristics of Petri-Nets were found by the authors to be strengths when using this particular modeling tool for simulations in the context of this research:

A MARKED PETRI-NET (TIME = 0)

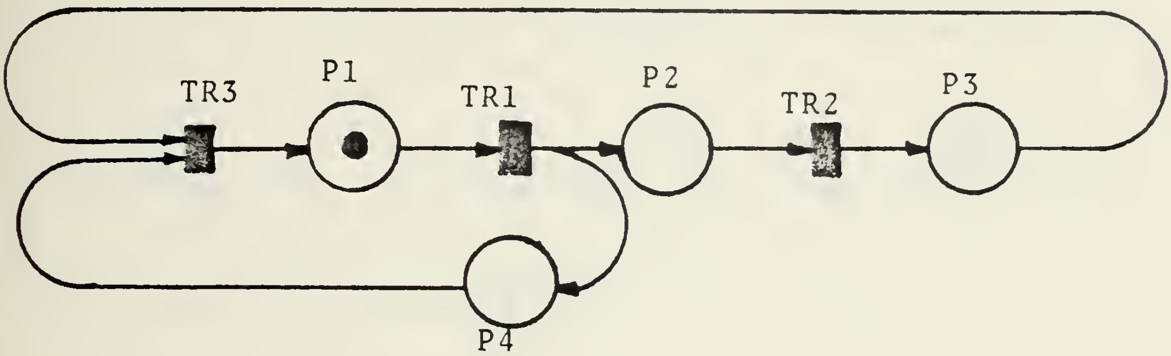

Figure 8

A MARKED PETRI-NET (TIME = 1)

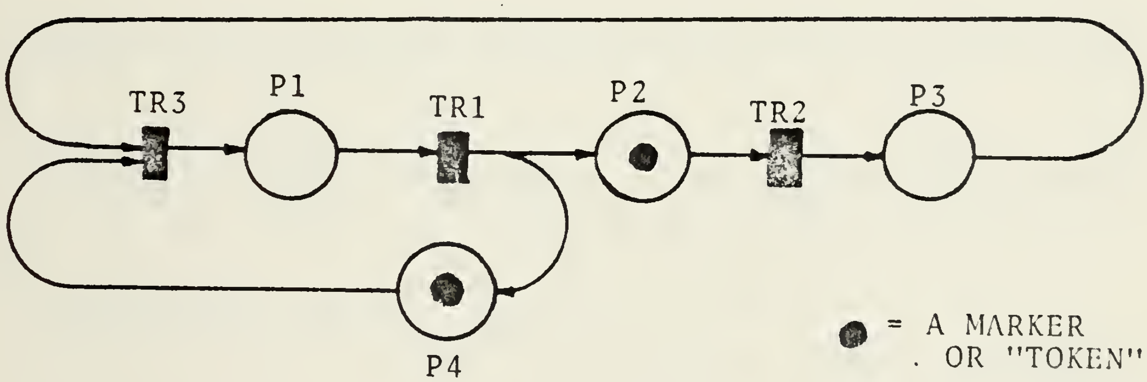

Figure 9

A MARKED PETRI-NET (TIME=2)

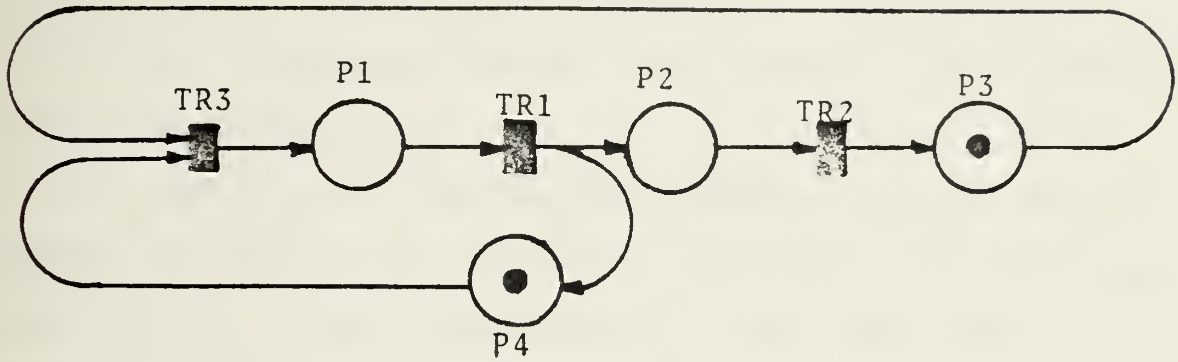

Figure 10

A MARKED PETRI-NET (TIME = 3)

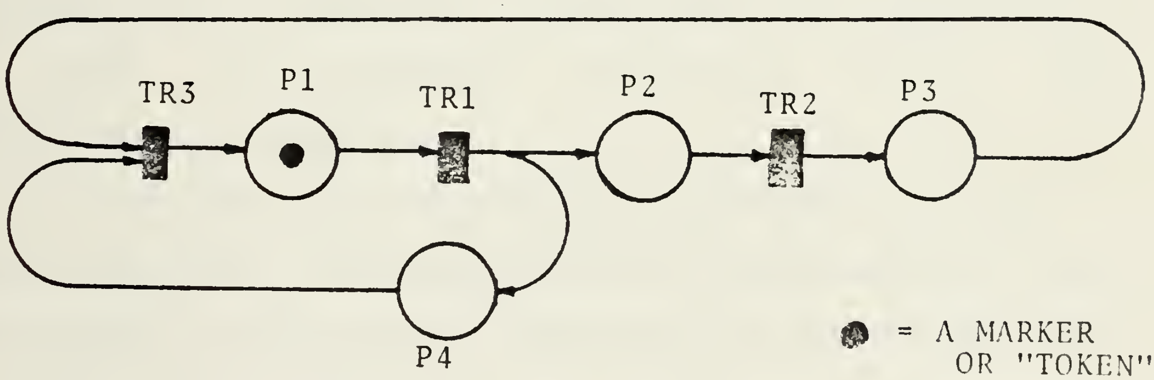

Figure 11

1. The rules governing Petri-Net execution are simple and easy to understand. This methodology can be quickly grasped by those with non-technical backgrounds who would ordinarily be unable to understand mathematical or analytical modeling. Yet Petri-Nets retain a high degree of precision and accuracy.

2. There is much flexibility inherent in the Petri-Net graph to model wide ranges of complexity. For instance, a model can be further abstracted by the replacement of a complex network of nodes by a single node.

3. There is a large degree of flexibility in assigning time intervals during execution.

4. Petri-Nets are well-suited to "snapshot" portrayal of network states. This advantage is important in simulation languages and is considered a strong point of languages such as GPSS and SIMSCRIPT. Petri-Nets possess this characteristic by nature.

5. Petri-Nets have the potential for a wide variety of uses. Basically, any process that can be flow-charted could be expressed by Petri-Nets. Applications could include: flow of information or control in an organization, information flow in electronics hardware, representation of computer software or, procedures and stages of development in a management program.

6. Petri-Nets lend themselves well to computer graphics display.

7. Petri-Nets are very effective when modeling concurrent, asynchronous activity in a network or system.

Certain weaknesses also became apparent to the authors in the course of this work. They are listed as follows:

1. Although Petri-Nets are basically simple to understand, the small building blocks of a network soon become exceedingly large and complex when large systems are modeled. The input files to some of the larger simulations in this paper were more than 1400 lines long. These networks must be drawn on paper before their entry into the computer, and this kind of effort soon becomes very tedious and prone to error.

2. Petri-Nets are best suited to concurrent, asynchronous behavior. When non-concurrent, synchronous behavior in a system is modeled the Petri-Net assumes a large amount of overhead.

3. The fact that Petri-Nets are not generally well known in the computer communications community could be a disadvantage when the user wishes to prove the accuracy of his model.

4. The fact that the simulator employed in this thesis effort was deterministic could be considered either as an advantage or disadvantage depending upon the application. Many simulations have value because of their stochastic nature. Certain classes of experiments, however, need to be understood not because of "chance" happenings but because of the operation of the "laws" of nature working upon the

elements of the experiment.

As with any modeling technique, success is achieved through the modeler's familiarity with the modeling tool. Petri-Nets provide an excellent means to model those applications best characterized by their asynchronous, concurrent nature.

V. PETRI-NET SIMULATIONS OF COMPUTER COMMUNICATION NETWORKS

A. INTRODUCTION

Once the basic rules of Petri-Net execution are understood, it is a simple matter to apply these rules to a communications network. The system modeled is the network itself. The movement of tokens in the Petri-Net represent the flow of information within the system: in this case, message traffic in the network. Each token becomes a discrete amount of information contained within the message.

The places in the Petri-Net graph are used to represent communications nodes within the network. The directed arcs of the Petri-Net graph are used to represent the communications links or channels which interconnect the nodes. The transitions between nodes indicate the availability of the channel to carry traffic. If the transition is enabled, the channel is clear, and the message is relayed from input node to output node.

The careful structuring of the Petri-Net graph imposes upon the modeled system a variety of network protocols. An advantage of using Petri-Nets for simulation purposes is that the logic and protocol of the system are entirely contained in the structure of the Petri-Net graph rather than in a complicated mathematical algorithm within simulation software.

B. TYPES OF NODES

The graphical output of the simulation attaches significance to the shape of the nodes displayed to the screen to facilitate recognition and interpretation. The experimental packet radio network in the San Francisco Bay area defines three primary types of nodes: terminals, stations, and repeaters. A terminal is a user node at which traffic is inputted or to which traffic is destined. It could be a fully automatic sensor, a handheld device, or a keyboard with CRT; but a terminal is a place where users connect to the network. The station is the node at which network control is exercised. The station typically keeps network statistics, monitors flow and congestion control, assigns routing, and performs data base management for the system. It is a terminal with increased processing capability usually provided by an attached mini-computer. The repeaters are stand-alone devices placed in numerous, dispersed positions throughout the network to act as relay sites. Repeaters do not act as origins or destinations of traffic, but they serve the purpose of extending the geographical range of the network beyond a typical line-of-sight distance.

While all networks do not use this identical terminology, these three functional nodes summarize the requirements of communication hardware in most networks. In the simulations in this thesis, three types of figures and labels represent the functions of nodes as described above.

C. A SIMPLE APPLICATION EXAMPLE

In order to understand more completely the application of Petri-Nets to communications circuits, refer to Figure 12. In this diagram two distinct one-way communications channels are represented. The first goes from T1 to T2 and the second goes from T3 to T5 and through T4. The individual tokens are representative of packets of information in a packet switched environment. The three additional nodes forming a triangle in the center of the diagram impose a special firing order upon the transitions in the communications channel. These center nodes are systems overhead which ensure that only one "packet" can be transmitted during a single time frame. In fact, if the three additional center nodes are thought of as a clock, then the entire network is a representation of Time Division Multiple Access in a network. A terminal can only transmit during a particular assigned time slot. After the time slot passes, the user must wait until the clock cycles back to his slot again. Because each transmitter has a unique time slot assignment, no two terminals can transmit during the same time, and collisions of packets on the radio broadcast channel are eliminated.

This explanation should give the reader a simple idea of the manner in which various protocols are represented. Obviously the Petri-Net in Figure 12 allows only non-concurrent activity on the communications channels.

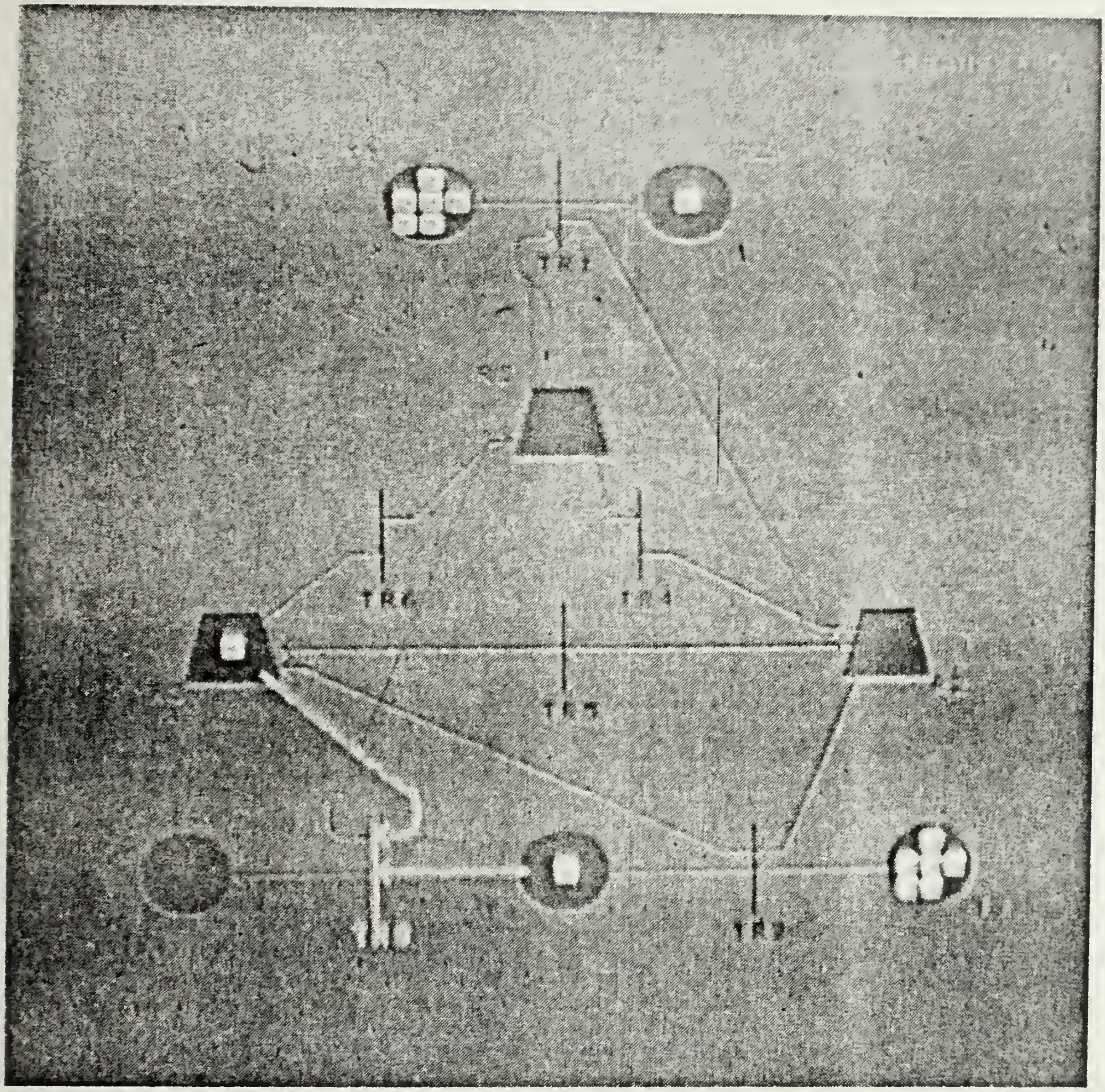

Figure 12.

The removal of the additional overhead would allow concurrent activity. In fact, this is sometimes desirable. For instance, those networks such as AUTODIN II which will make use of leased landlines will allow concurrent activity in the network. The Petri-Net is more efficient when modeling concurrent activity. The requirements to ensure non-concurrency as in Figure 12 , causes one additional node and one additional transition to be placed in the Petri-Net graph for every transition on the communication links. Using the fundamental Petri-Net described here as a small building block, a system of considerable complexity can be built with many origin and destination terminals, and which allows packets to flow two directions with multiple, alternate paths from source to destination.

D. RANDOMNESS IN PETRI-NETS

The Petri-Net simulator used in this thesis work is deterministic. After the simulator begins execution there is no means to interactively alter the sequence of events, and there is no element of randomness within the simulator. Because of this situation, the same input file will always give identical output. Although the capacity for alternate routing within the communications network is implemented, the tokens do not randomly "choose" their routes during execution. They can only follow their pre-assigned routes from origin to destination. This means effectively that fixed routing instead of adaptive routing is represented in the simulations. This is not necessarily a disadvantage,

however, as some simulations have shown better throughput and time of delivery results using fixed routing over adaptive routing [20].

It might be advantageous to modify the Petri-Net simulator to make it more stochastic in character. This might be accomplished in at least two ways. First, the inital marking of the network could be varied randomly at the beginning of each run. Certain key nodes could be marked or left unmarked according to the result of a call to some random number generator. Then the initial random state of the network would affect the end result of the output file.

A second way to add randomness deals with the Petri-Net concept of "dynamic conflicts" (see Figure 13, for an example of this particular network state). In this figure, the reader will notice that both Transitions 1 and 2 are enabled, but both cannot fire. Only one transition can fire, since in so doing it removes the token from T2 and disables the other transition. Thus TR1 and TR2 are said to be in conflict. This basic relationship can be used to create either deterministic or nondeterministic behavior. If the Petri-Net simulator is deterministic, the firing order for transitions in conflict is fixed according to a certain rule. This describes the case in this paper. The firing order of transitions is explicitly defined by their ordering in the input file. In the case of Figure 13, if TR1 is listed in the input file before TR2, and if a dynamic conflict occurs between them, TR1 will always fire first and

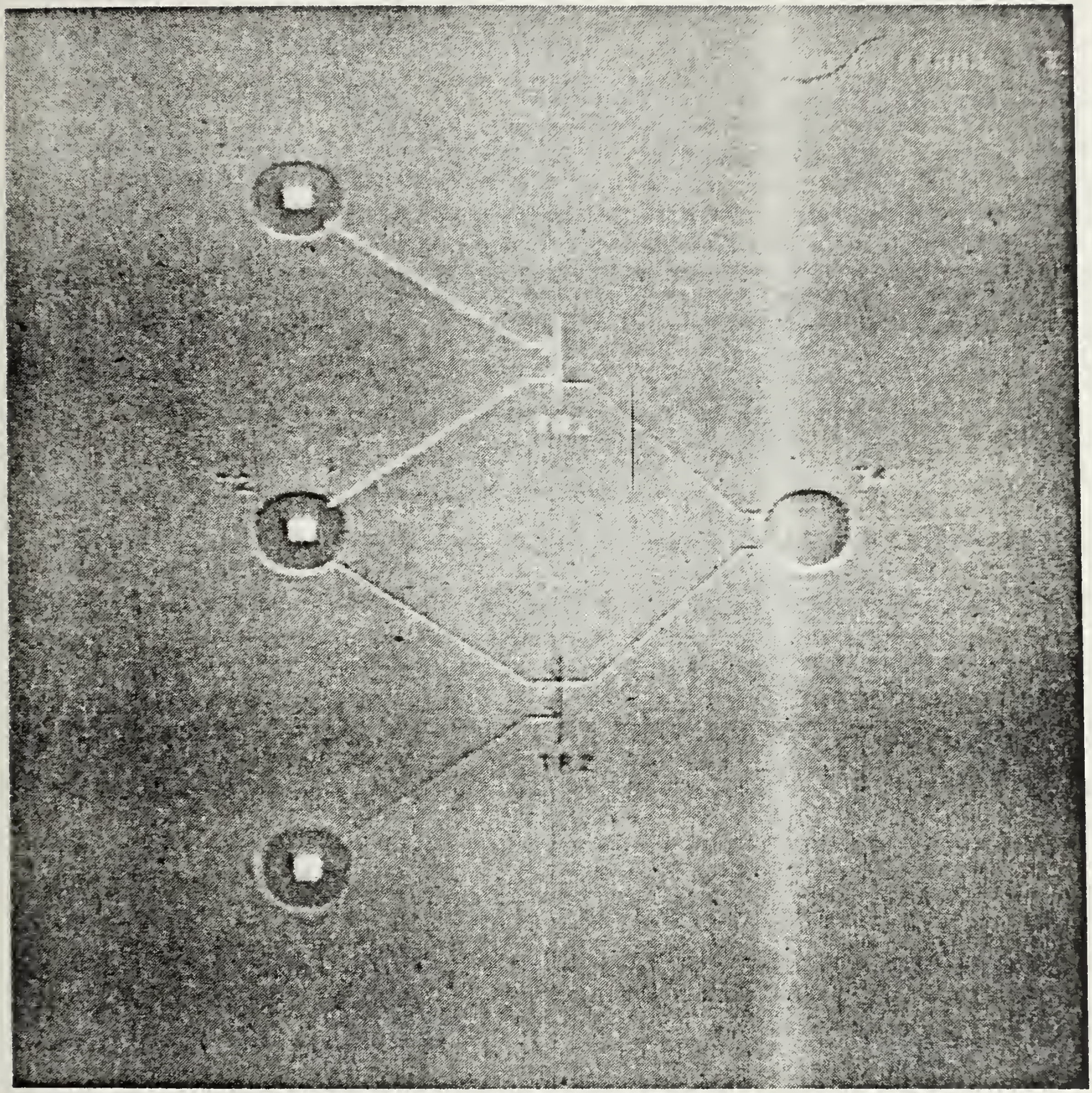

Figure 13.

TR2 will not fire during that time interval because it will have been disabled. This set of rules gives a strict priority of firing to the network.

The firing order of transitions in conflict could be modified so that it occurred in a random fashion. This would allow the Petri-Net to be executed in a non-deterministic manner and add the missing aspect of randomness to the outcome if so desired. In fact, Petri's first networks were non-determinestic because of this factor.

E. MEMORY STORAGE REPRESENTATION

The amount of memory storage in any particular communications node is also easily represented in the simulation. In this work a maximum number of seven packets is allowed at any one node. This number reflects the buffer size of seven packets in packet radio technology. If at any time a node accumulates more than seven packets, the buffer size has been exceeded and packets would theoretically be lost. When the buffer space is exceeded, the number of overflow packets is displayed outside the node in a red warning color.

F. SYSTEM LOAD AVERAGE

The system load on the network can be represented by the number of packets in transit at any one time interval. The system load then can be varied by controling the frequency of message generation. The shorter the interarrival time between generated messages, the busier the network will be.

At some point the network will be saturated and unacceptably congested if message input is greater than message throughput.

It is a simple matter to construct a message generator using Petri-Nets. See Figure 14. In this figure Transition 1 is firing to two outputs. One can be thought of as external and one output, R1, can be considered internal to the network. The external node can represent entry into the communications channel. The internal output feeds a token back into the "generator" and will, therefore, fire other packets into the network at regular intervals. The other repeaters (R2, R3) can be thought of as delays which slow down the frequency of message generation. This configuration of places and transitions constitute a message generator. The frequency of generation can be staggered and then several generators may be placed at the input of every communications circuit. In this manner the system load on the network may be varied.

G. TIME REPRESENTATION

The Petri-net model is very good at representing the net status at distinct time periods. In fact, each time interval displayed to the screen gives an excellent "snapshot" of network status. This is an important advantage inherent to the Petri-Net simulation. Another advantage is the flexibility afforded in assigning the time interval. The user has the prerogative of making each time interval as long or as short as is necessary. Successive snapshots may

represent the passage of time of 1 millisecond, 100 milliseconds, or 1 hour, depending upon the application and network being modeled.

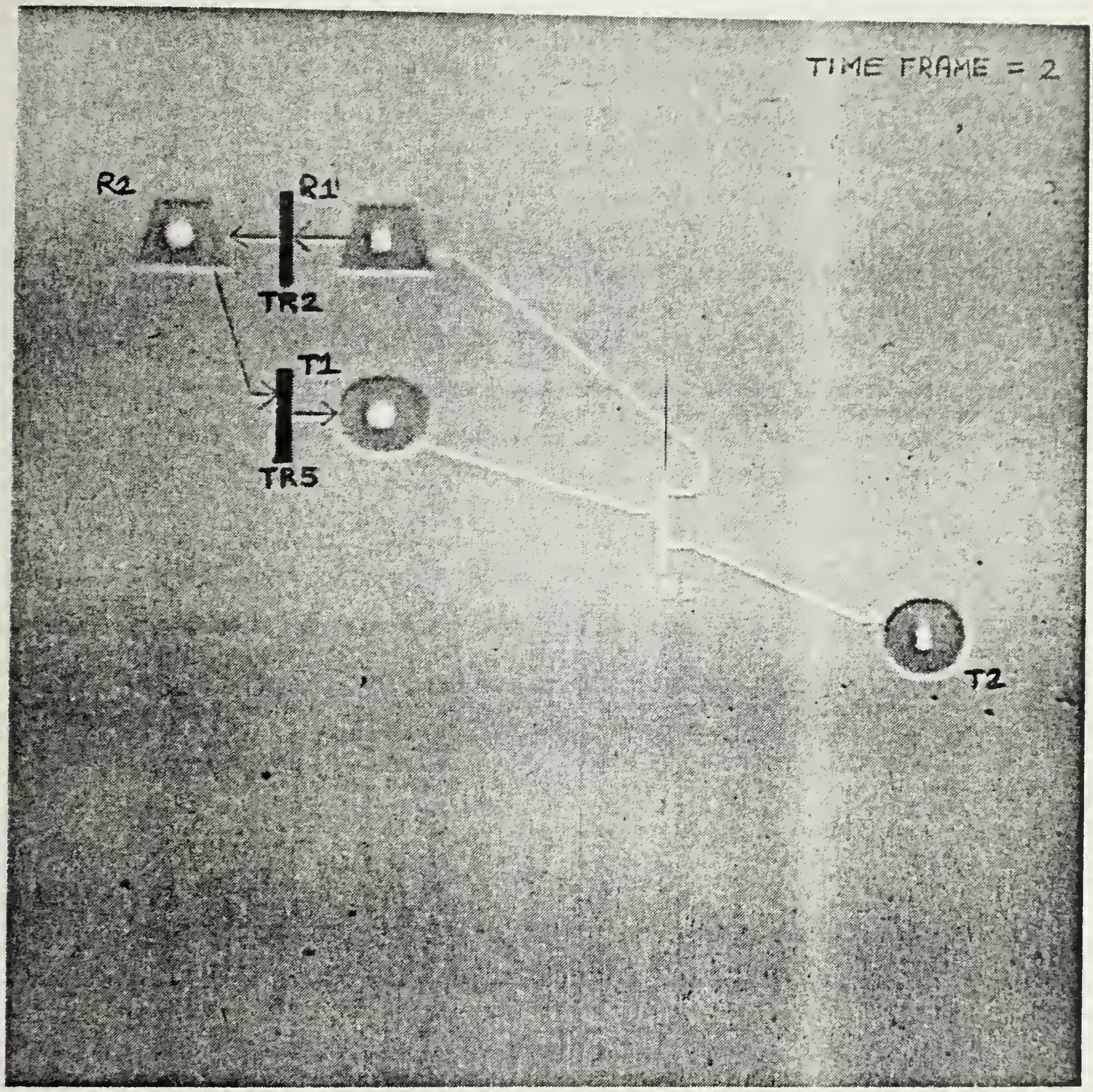

Figure 14.

VI. THE EXPERIMENT

A. DESCRIPTION

An experiment was designed to demonstrate the feasibility of using the Petri-Net simulator as a predictor of network performance. The experiment was performed by keeping certain parameters constant and varying others. A series of six input files were run through the simulator. In all six files the network architecture and the fixed routes were kept constant. Figure 15 shows the network. There were five origins of message traffic and five destinations. Four nodes were designated as terminals plus one station. There were four repeaters that performed relay functions within the network. Each of the five origins could send traffic to one of four destinations. Each source to destination combination had two routes for traffic to take. This made a total of 38 possible fixed routes in the network.

The controlled variables for the experiment were system load, concurrent vs. non-concurrent activity, and polling frequency of various circuits. The first three runs of the experiment were done with a high load. For the second three runs the message generators were slowed down to give a low system load. Some input files allowed concurrent network activity and some required non-concurrent activity. On two of the non-current runs the frequency of polling certain selected circuits was increased. This would represent the

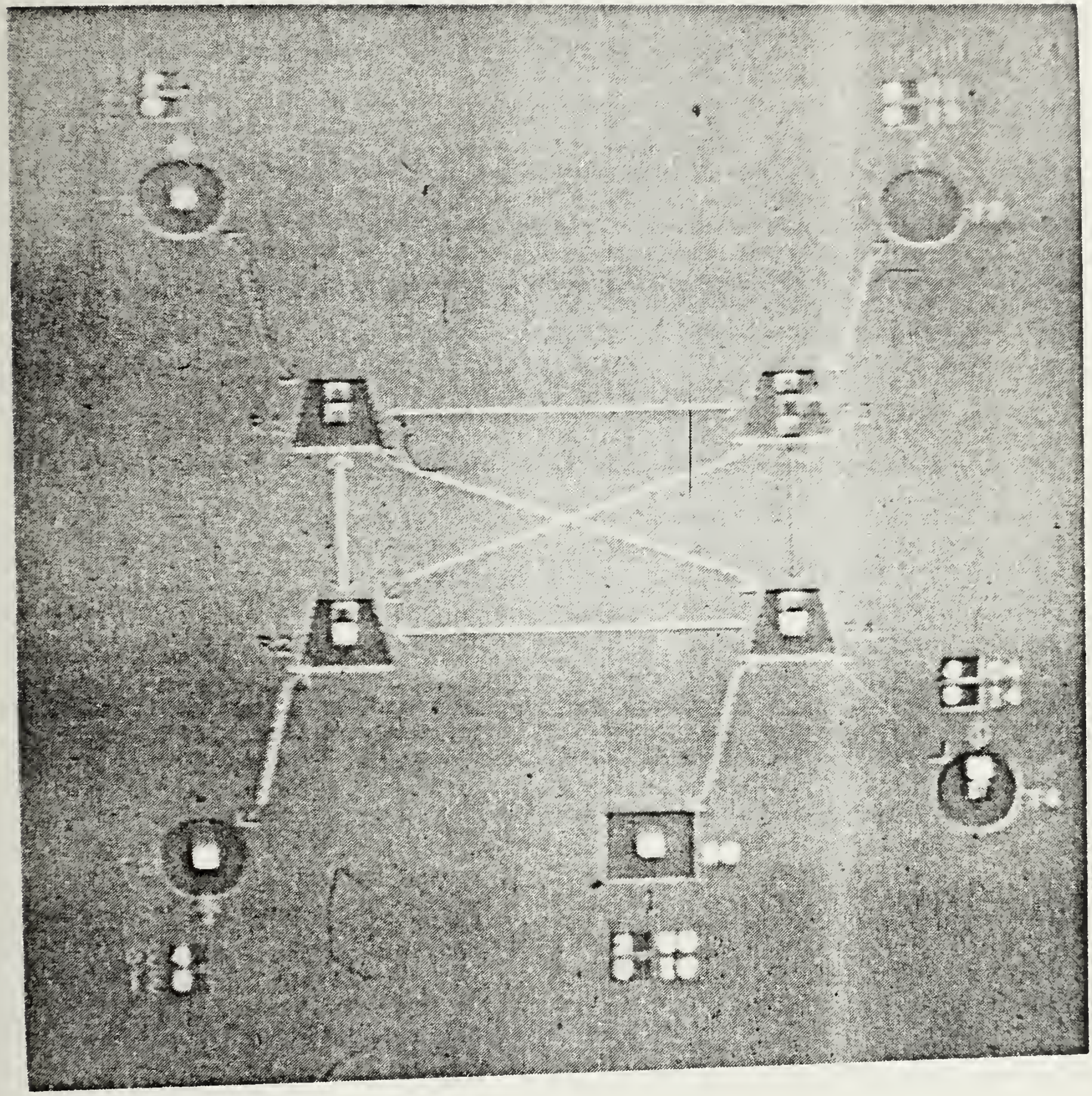

Figure 15.

equivalent of assigning a certain user more time slots in a TDMA scheme than another. It would give pricrity to those subscribers who have more traffic to send.

The only statistic gathered from the experiment was throughput measured in the number of packets which successfully reached their destination. This number could be extracted directly from the output queues at each terminal or station. The following points sumarize the characteristics of each of the six input files:

1. Runs 22-24 were run at high load.

2. Runs 25-27 were run at low load.

3. Runs 22 and 25 exhibit concurrent network activity.

4. Runs 23 and 26 exhibit non-concurrent network activity with equal polling frequency of all circuits.

5. Runs 24 and 27 exhibit non-concurrent network activity with weighted polling on certain circuits that are terminated at T1.

B. RESULTS

Figure 16 shows a summary of throughput statistics from the experiment. Certain results are no doubt obvious, but the quantitative nature of output statistics validates prior assumptions.

The following obervations are noteworthy;

1. Allowing concurrent activity on the network greatly increases throughput. The number of packets successfully transmitted in RUNS 22 and 25 was on the order of four times larger than the non-current runs. This question of

THROUGHPUT IN NUMBER OF PACKETS

HIGH LOAD

	RUN22	RUN23	RUN24
T1	92	16	25
T2	62	16	15
T3	77	18	16
T4	58	18	17
S1	62	17	16
TOTAL	352	85	89

LOW LOAD

	RUN25	RUN26	RUN27
T1	36	17	19
T2	38	17	16
T3	35	18	16
T4	33	16	15
S1	31	17	16
TOTAL	173	85	82

Figure 16.

concurrency has significant implications to radio broadcast systems. For instance, what would be the advantages gained in the Packet Radio Network if terminal radio output power were reduced so that the terminal could only talk to its nearest neighbor rather than the entire network? This situation could allow simultaneous transmission of packets without the threat of collision. Also, a multiplexing scheme within the network could allow concurrent activity. These types of considerations could be modeled easily with the appropriate modifications to the Petri-Net graph.

2. Increasing the frequency of polling on selected circuits increases the throughput of those circuits. The reader should note the number of packets received at T1 in RUNS 23, 24, 26, and 27. RUNS 24 and 27 show a slight increase because certain circuits destined for terminal T1 were polled more frequently. This situation could easily reflect the assignment of more time slots to certain priority subscribers in a TDMA scheme. Again this modification was performed simply by restructuring the input file to the Petri-Net simulator.

3. The reader will note that the total throughput in the non-concurrent runs is largely the same regardless of the high load - low load factor. This is because the system is basically "saturated" during non-concurrent activity at both high and low loading. Although the user might desire to improve throughput by generating more messages and trying to force them into the system, the network will give the same

results because it is already operating at full capacity.

4. A visual inspection of the output as RUN22 executes reveals that the system is essentially operating at peak efficiency under maximum load. The viewer will notice that the buffers at every location are frequently filled to capacity but seldom are overflowing.

VII. RECOMMENDATIONS AND CONCLUSIONS

A. RECOMMENDATIONS

The potential exists for significant follow-on work to this thesis. Major topics for future work include the following:

1. The development of a language to describe networks, and the inclusion in the software of a "front-end" program that would make the input file less cumbersome. The program could be static, like a compiler, or interactive, written to query the user about a number of basic network parameters. For instance: How many nodes do you desire in the network? What are the paths of traffic from source to destination? Do you wish to allow concurrent or non-current activity? The software would take the user responses to such parameters and construct the input file from the responses.

2. A statistics gathering package should be written to collect and collate vital network parameters as the simulator is executed. Such a package would keep a running total of such items as:

a. average number of packets at a node

b. average number of packets on a circuit

c. percent use of a circuit

d. average time delay between transmission and reception along each fixed route

e. number of messages lost

f. number of messages which successfully reach destination.

These statistics could be formatted and displayed by a post-processor.

3. As previously described, the addition of randomness to the simulator might be considered desirable, and , if so, the proper modifications could be added.

4. As the system approaches "production status", the entire question of model validation needs to be addressed carefully. Much could and indeed has been written on the subject of how to demonstrate your simulation is accurate. [10] As in most simulations there is a tradeoff between the degree of complexity represented in the model versus the largeness of the network. If a network is small or if only a small segment of the network is modeled, the degree of detail can be great. If the network is very large, however, the data storage capability might not allow the same level of detail.

There were two constraints to validation posed in this work. First, the memory capacity of the PDP-11/70 was stretched to the limit on the larger runs. The mini-computer offers 128,000 bytes of memory which are partitioned into thirds. This gives any one user 128/3 or 42.6 K bytes of memory. The graphics programs approached and then exceeded this bound before the work was completed. This forced the division of one program into two separate programs of 38K each. The file sizes for the output files from the Petri-Net

simulator are also very large. The larger of the input files produced output files on the order of 300K bytes. Figure 17. shows the relationship between number of nodes in the Petri-Net versus the size of the output files. In order to make claims of simulation accuracy based on real world networks, more memory is needed.

It should be added at this point that limitation of memory caused changes in the graphics display programs and in the overall organization of the software. The potential exists in the actual Petri-Net simulator (written in fortran and discussed in Appendix A) for the execution of Petri-Nets of well over a thousand places. If the user does not require a graphics output from this software package, then larger networks can be simulated on the PDP 11/70.

Secondly, the networks that were of most interest to the authors are largely experimental, unproven technologies. The Autodin II network is not yet operational and no statistics are available for validation purposes. The JTIDS technology is likewise not established operationally and many of the system characteristics are classified. Packet radio, which formed a good deal of background for the simulation, is still in its infancy. Also, packet radio employs a random channel access scheme which is contrary to the deterministic nature of the Petri-Net simulator.

Because these are new technologies, new routing and flow control algorithms and a host of different kinds of protocol are presently being developed. It is difficult to model a

system that has not reached its production state. Rather than being concerned about modeling a particular network that is still in a process of change and then trying to prove the simulation valid, it seemed wiser to leave the models in a more general state. By executing a variety of different input files, the simulations demonstrate the feasibility of future validation.

B. CONCLUSIONS

The conclusions of the authors are fourfold:

1. First, computer communication networks can be meaningfully modeled with the use of Petri-Nets. The background research to this thesis discovered no previous work which employs Petri-nets in the manner described in this paper.

2. Secondly, Petri-Net models of networks can be executed and displayed effectively on a color graphics terminal. The results of such a simulation are more easily understood than the common, hard-copy outputs produced by most analytical or queuing theory simulations. The color graphics output also could have considerable educational value. Again, background research uncovered no instance in which Petri-Nets were displayed and executed on a color graphics terminal.

3. Thirdly, and perhaps most importantly, the implementation of such a modeling technique in a production environment as a predictor of system performance appears feasible.

4. Fourthly, there appears to be considerable benefits to encouraging future, carry-on work in the subject matter of this thesis.

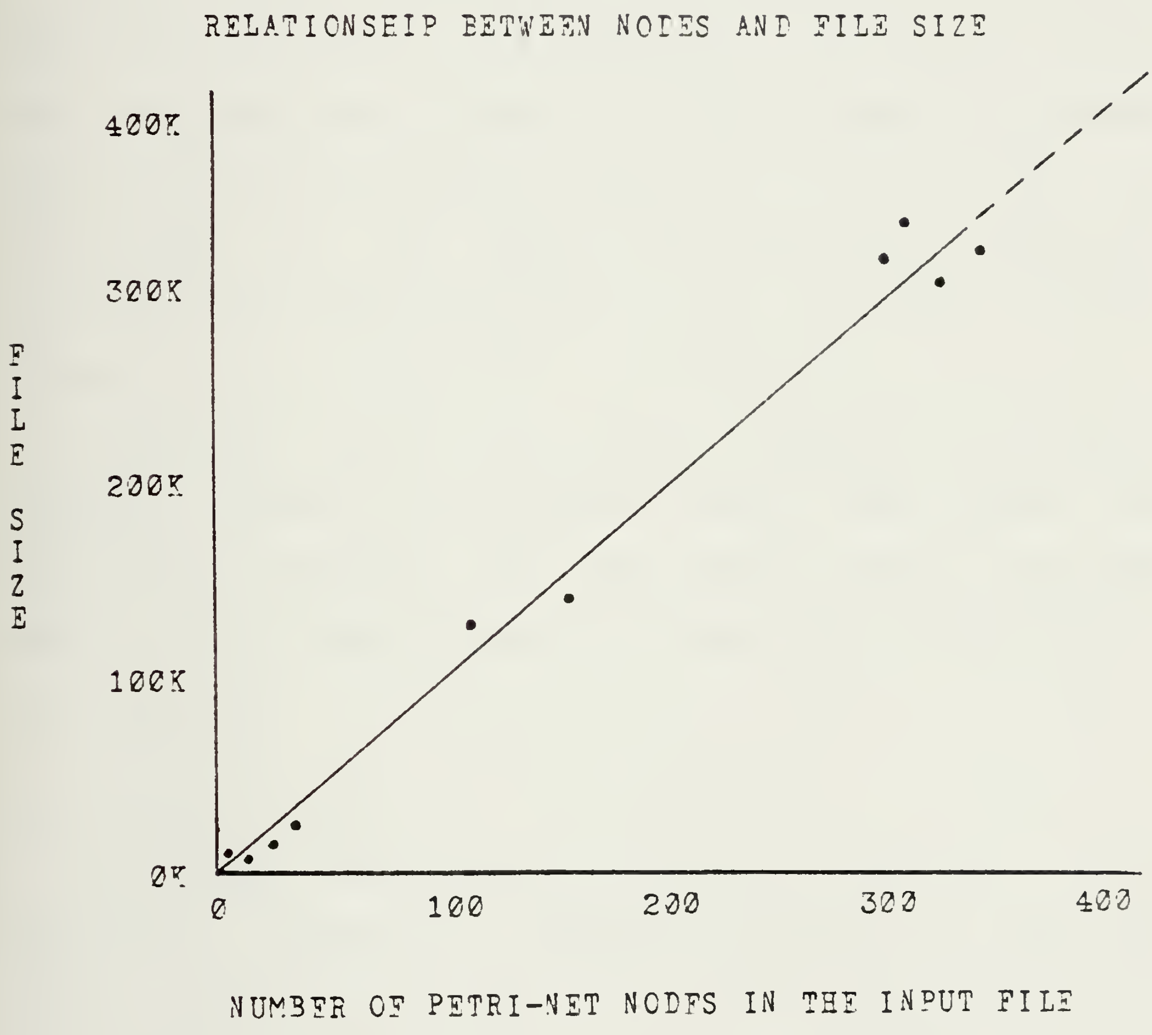

Figure 17.

APPENDIX A - USER INSTRUCTIONS FOR THE PETRI-NET SOFTWARE

A. INTRODUCTION

This chapter is written to describe certain procedures and syntax peculiar to the simulation software. Assuming that an interested student or faculty member is somewhat familiar with the theory and structure of Petri-Nets, the instructions in this section will allow him to apply the simulation and graphics output to his particular modeling problem.

Figure 18 shows the various components of the entire software package, the program source code sizes, the programming language, and the output files. The reader should refer to this figure as he reads the instructions in this appendix.

B. THE INPUT FILE

The input file written by the user contains all of the information necessary to uniquely describe the Petri-Net model. This file is read by the fortran f4p program named. "simulator". The input file must be named "RUNXX, where XX can be any number from 01 through 99. There are three main divisions of the file,

Part I   Places

Part II   Transitions

Part III   Marking

SOFTWARE METHODOLOGY

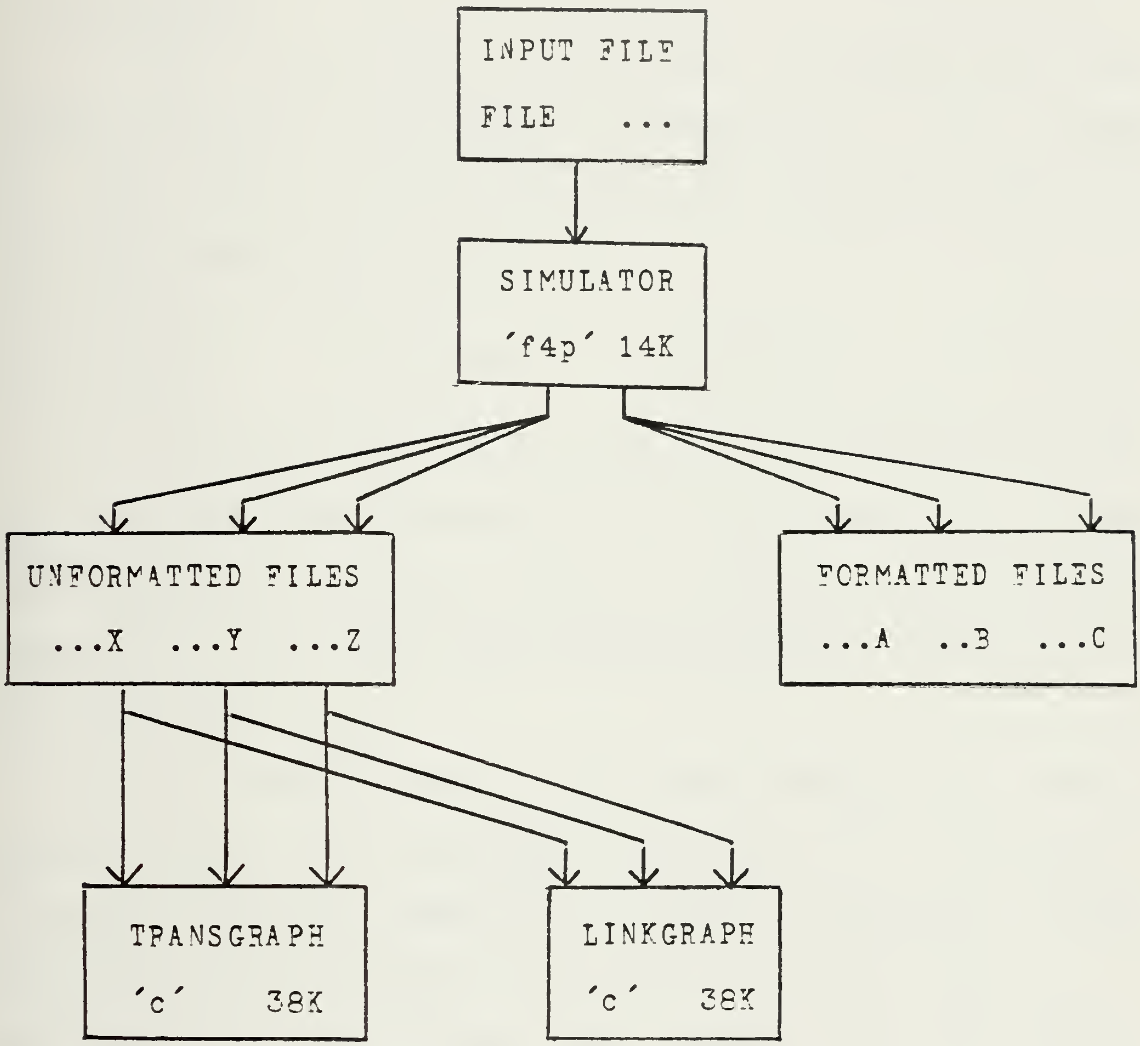

Figure 18.

Appendix B. shows a sample input file. The reader should refer to this example as he reads the following instructions.

1. Places

The first line of the file specifies the total number of places to be read in. Beginning on line number two, the names and locations of the places are listed. The following format applies:

Place Name .... X-cordinate .... Y-cordinate .... Plot Flag

The place name must be less than 13 characters long. Only the first two characters will be displayed as a label on the graphics terminal. There are unique instructions for labeling places in the multi-routing version. These will be discussed later.

The first letter of the name specifies the type of figure that will be displayed. The letter, "T", identifies a "terminal" and will appear as a circle on the output screen. The letter "S" identifies a "station" and will appear as a rectangle on the screen. The letter "R" identifies a "repeater" and will be displayed as a truncated triangle. Input and output queues can be represented by placing an "I" or "O" as the first character of the place name. These nodes are displayed as small rectangles, large enough to contain a two digit number. Names may begin with letters other than those listed above. They will not, however, be displayed to the screen.

After the name is listed, on the same line, the screen location of the place is specified by means of an x, y coordinate system. The x and y values are in the range of Ø-511, with the (Ø,Ø) point located at the top, left-hand corner of the graphics display unit.

The third item of information in the "places" line is a "plot on/off" entry. The user will frequently have places identified in the Petri-Net which are necessary control elements, but which do not need to be displayed on the output. A value of "1" will cause the place to be displayed with its label. A value of "Ø" is used for listing places which are not displayed to the output screen.

2. Transitions

After every place in the Petri-Net is listed (one line per place), the transitions are listed. As before, the first line specifies the total number of transitions to be read in.

Then, the transitions are each listed in a three-line format as follows:

Transition Name...X Coordinate...Y Coordinate...Plot Flag
Places into Transition
Places out of Transition

Transitions are named "TRXX", followed by their x, y coordinates, and a "1" or "Ø" to indicate whether they are to be displayed to the screen. The second line concerns the

input to the transition.

The first field of this line specifies the total number of inputs, and the following numbers indicate which places enter that transition. The numbers,

3 8 11 14

for instance, indicate that three places are inputs to this transition. The particular places are identified by their implicit, line-number ordering as entered in the list of places. In this example the three places are the 8th, 11th, and 14th places entered in the input file.

The third line of the transition entry concerns the outputs from that transition. The format is identical to that of the line above, i.e., the numbers

2 3 4

indicate that the transition fires to two outputs, the 3rd and 4th places listed in part one of the input file.

3. Initial Marking

After all the places and transitions are listed, the Petri-Net must be given its initial state of marking. The initial placement of tokens is specified by the following format:

MARK Line # of Place # of Tokens

Thus, the entry "MARK 3 1" specifies that at the beginning of the simulation, Place 3 (the third entry in the list of places) is marked with one token. Several places may

be marked initially, but each marking requires a separate line.

Every line in the input file begins in the first column. Do not indent the beginning character of the line. The fields within each line must be separated by one (or more) blank spaces. The final line of the input file is the command "END".

4. Execution

When the user completes the input file he should exit from the edit mode and is now ready to execute the program "simulator" by typing "simulator.out". The program will ask him which input file he wishes to read, and the user responds by typing "RUNXX" (the file he previously created) and a <cr>.

C. THE OUTPUT FILES

The program "simulator" produces six separate output files. When RUN01 is entered into "simulator", files named RUN01A, RUN01B, RUN01X, RUN01Y, and RUN01Z are produced. The files suffixed with A through C are formatted files. Files X through Z are unformatted. Files A and X contain the essential data structures that have been read in "simulator" from the input file. Files B and Y contain the markings for each place at successive time frames which will appear on the graphics output. Files C and Z contain information concerning which links or transitions fire at any particular time frame and are used to highlight present activity on the screen. The graphics display programs, written in "C",

require unformatted input files. The formatted files are necessary for the user to validate that correct input data is reaching the graphics program, and to troubleshoot when locating a problem. Examples of these files are contained in Appendices C through E.

D. USER OPTIONS

1. Choice of Programs

After the program "simulator" has executed the input file, the "C" programs read the output files and display the results of the simulation to the screen. At this point the user has several options concerning the method of display.

There are two separate programs the user can select--"transgraph" and "linkgraph". By executing "transgraph" the viewer will be able to observe the nodes of the network together with their associated transitions. "Linkgraph" does not display the transitions, but links node-to-node in the common way that communications networks are most frequently represented. For simple networks or to explain the basic working of Petri-Nets, the user will probably desire to see the transitions. "Transgraph" will only run with less than 100 places. For more intricate networks that contain several hundred places and transitions, the "linknode" program is necessary to avoid congestion on the screen.

2. User Questions and Responses

After selecting which program to run, the user is given a series of questions from the CRT. Question one asks

the user to select which input file he wishes to execute. He responds with the command RUN X. For instance, if the input file named RUN21 has been executed by "simulator", and the user wishes to see the results, he enters RUN21X in response to question one. (Numerous files may be waiting in the users' directory which can be called in for display.)

Question 2 asks the user if he wants to view the data structures of the program. By entering a "1", the data structures will be printed to the CRT. A "0" will move on to the next question without viewing the data structures.

Question 3 asks the user which of three versions of the program he wishes to see. Version 1 displays the marking of tokens in the conventional Petri-Net fashion, with numbers printed at the center of the places. Version 2 represents tokens by single, yellow boxes printed inside the nodes. These boxes are designed to represent "packets" of information in the packet switching concept. As previously described, each node has the capacity to hold seven packets. If the number of packets goes over seven, a red overflow number appears beside the saturated place.

The third version is the multi-routing, multi-destination version. Version 3 uses color in a unique way. Because packets may be originating at different nodes and traveling to several destinations, the linking channels require two-way transmission. The graphics display is color-coded to highlight this information. Packets traveling to a particular place are colored to match the label of that

place. For instance, when a green box appears in the network, the viewer can trace its progress to the destination whose label is displayed in green. In Version 3 (multi-routing), a packet originating at a certain node and destined for another particular node may take different paths to arrive at the destination.

The user at this point must select one of the three versions by entering a "1", "2", or "3" followed by a <cr>.

Question 4 asks the user to select one of the three Genisco graphics terminals in the C3 lab on which the output will appear. Correct entries to this query are "Ø", "1", or "2".

The fifth question asks the user if he desires a time pause of two seconds duration between time frames of the simulation. If he desires the capability to look closely at each network snapshot, he enters a "1". If not, he should enter a "Ø" and the simulation will run without pauses. This gives the viewer more of a "real-time" impression.

At the end of this final question, the screen displays the initial condition of the network. After noting the initial condition, the user should type a carriage return to continue execution.

If the user has typed a "1" in response to question 5, he also has the ability to indefinitely suspend execution of the program at any time frame. He can interrupt the program by typing a "BRK" from the CRT keyboard. After studying that particular snapshot of network status, he may

continue normal execution by typing a carriage return.

This interrupt does not work if the program is executing without pauses. Typing two consecutive "BRK's" will enable the user to exit the program entirely.

3. The "Highlighting" Feature

The linking algorithm in both "transgraph" and "linkgraph" draws lines in a dark blue color. The highlighting feature in both programs changes the blue connecting link color to bright yellow on those links which are carrying traffic at any particular time frame. This feature performs in the following sequence. (1) The link lights up at the point where the future action will occur. (2) The packet(s) in question moves from one end of the highlighted link to the other. (3) The highlight remains on the link to emphasize where the action occurred. The user will notice that direction of movement on the highlighted link is indicated by an arrow pointing in the appropriate direction. See Figures 19 and 20.

E. UNIQUE INSTRUCTIONS FOR VERSION 3:

In order for the user to implement the capabilities of Version 3, special network design information must be included in the input file. This paragraph describes these special instructions.

The simulation is structured to function in a fixed routing manner. That is, the multi-routed paths are predefined by the user.

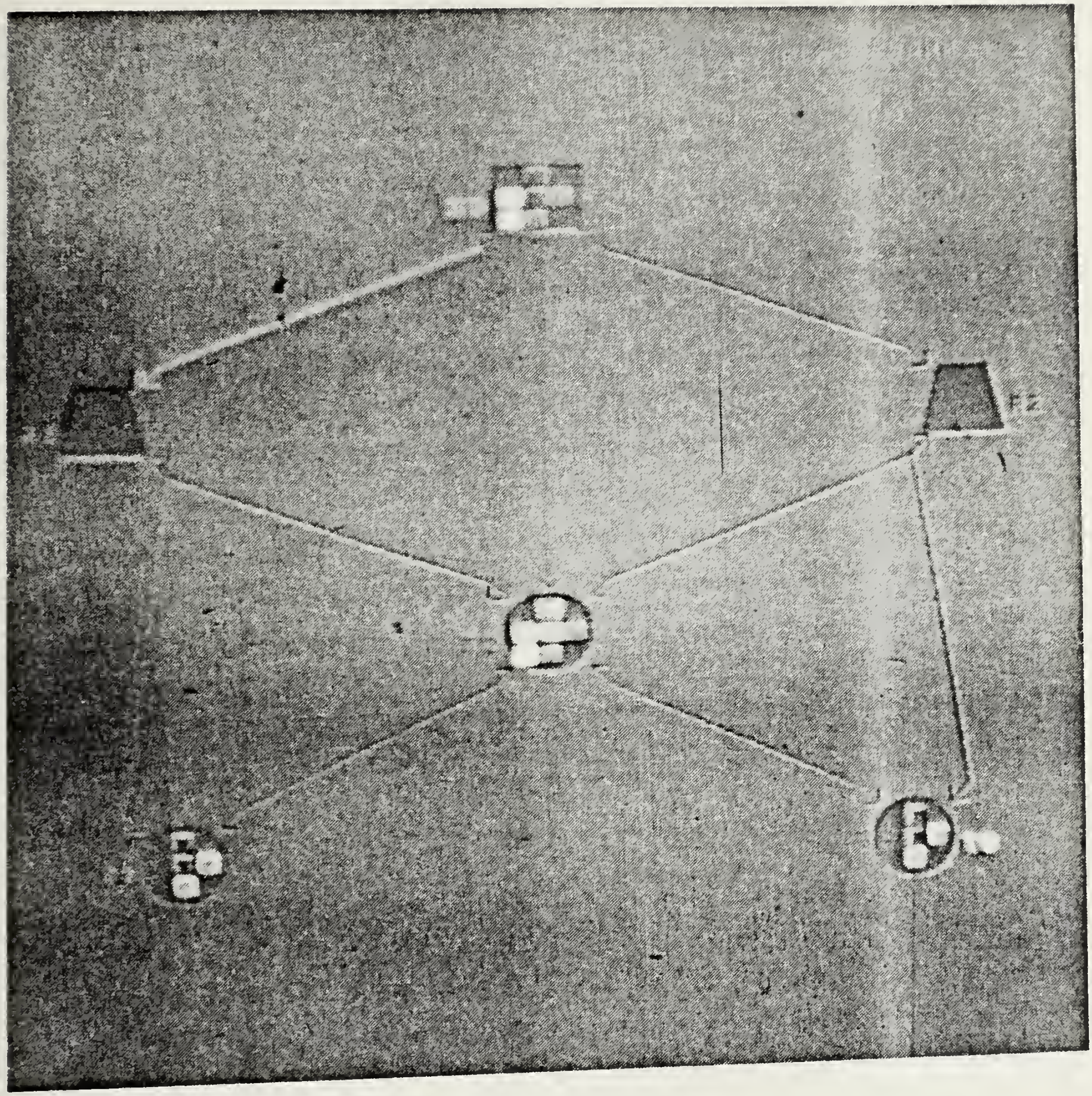

Figure 19.

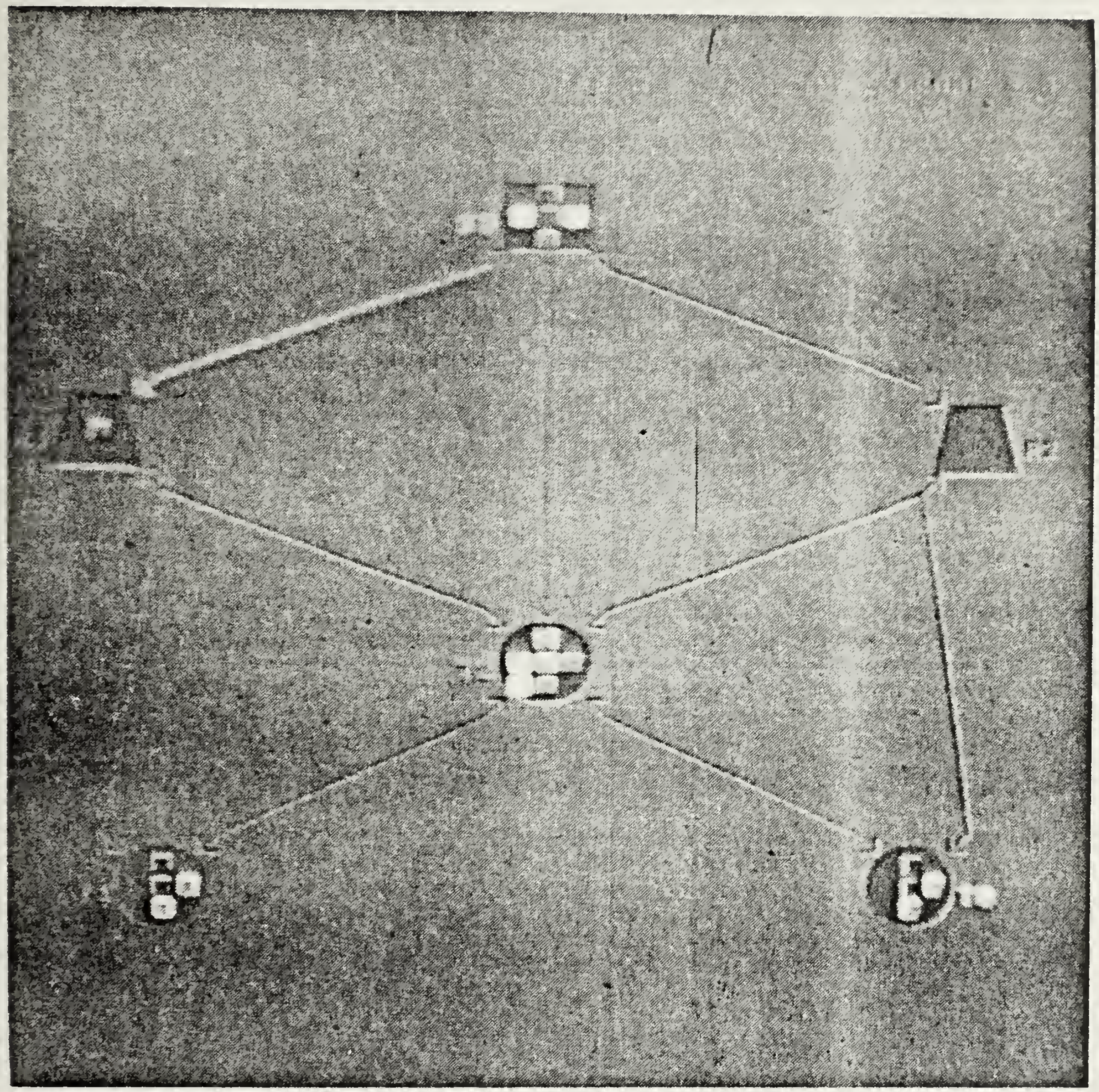

Figure 20.

There can be as many alternate paths from source to destination as the user wants to include. But once these paths are specified in the input file, they do not change dynamically during execution.

The multi-routing feature is made possible by "stacking" numerous places on top of one another and displaying these piggy-backed places at the same coordinates. In this manner the nodes appear on the screen as a single place, although in reality they may be buried several deep.

As the total network is conceptualized by the user, he must begin by mapping out all origins, all destinations and all relay nodes. Every node becomes unique to a particular path. For instance, a simple case would be to send a packet from T9 to T5 (see Figure 21.) by two different routes--Route 1 goes through R1 and Route 2 goes through R2.

In this case T9 and T5 would be "stacked" two deep. The routes are T9-R1-T5 and T9-R2-T5. Because they are plotted at the same point, the terminals appear as a single node.

The "header" information to perform this routing is contained in the name of the place. In Version 3, every node that will be displayed to the screen must be assigned a seven unit name in the input file.

The first unit of the name specifies the type of figure to be displayed ("T", "S", "R", "I", "O" as previously explained). The second unit specifies the color that the name will be printed in. This number is derived from the particular color table (16 colors are in a color table)

that is being used by the graphics program. The color of the label must correspond to the color of the packets bound for that route's destination. Units 3 and 4 of the name designate the route number that the node lies upon. Route numbers are arbitarily given by the user and utilized for his own identification purposes. Unit 5 designates the color of the packet that will travel along that particular route. Every node on that route have the same color designator. The packet color of the route is determined by the destination of that route. Units 6 and 7 of the name specify how many nodes are stacked at that location. Places that are stacked must be listed together in the input file.

An example of this 7 unit name might appear as follows:

T205419

Field 1 designates that a circle will be drawn.

Field 2 specifies that the label will be displayed in color number 2 of the program's color table.

Fields 3 and 4 show the place on route number 05.

Field 5 ensures that every packet which passes through this place will be displayed in color number 4.

Fields 6 and 7 specify that places are stacked 19 deep at this coordinate.

As the program executes, the stacking algorithm totals all the packets which are located at a particular stacked location and displays that total number of boxes in the appropriate colors.

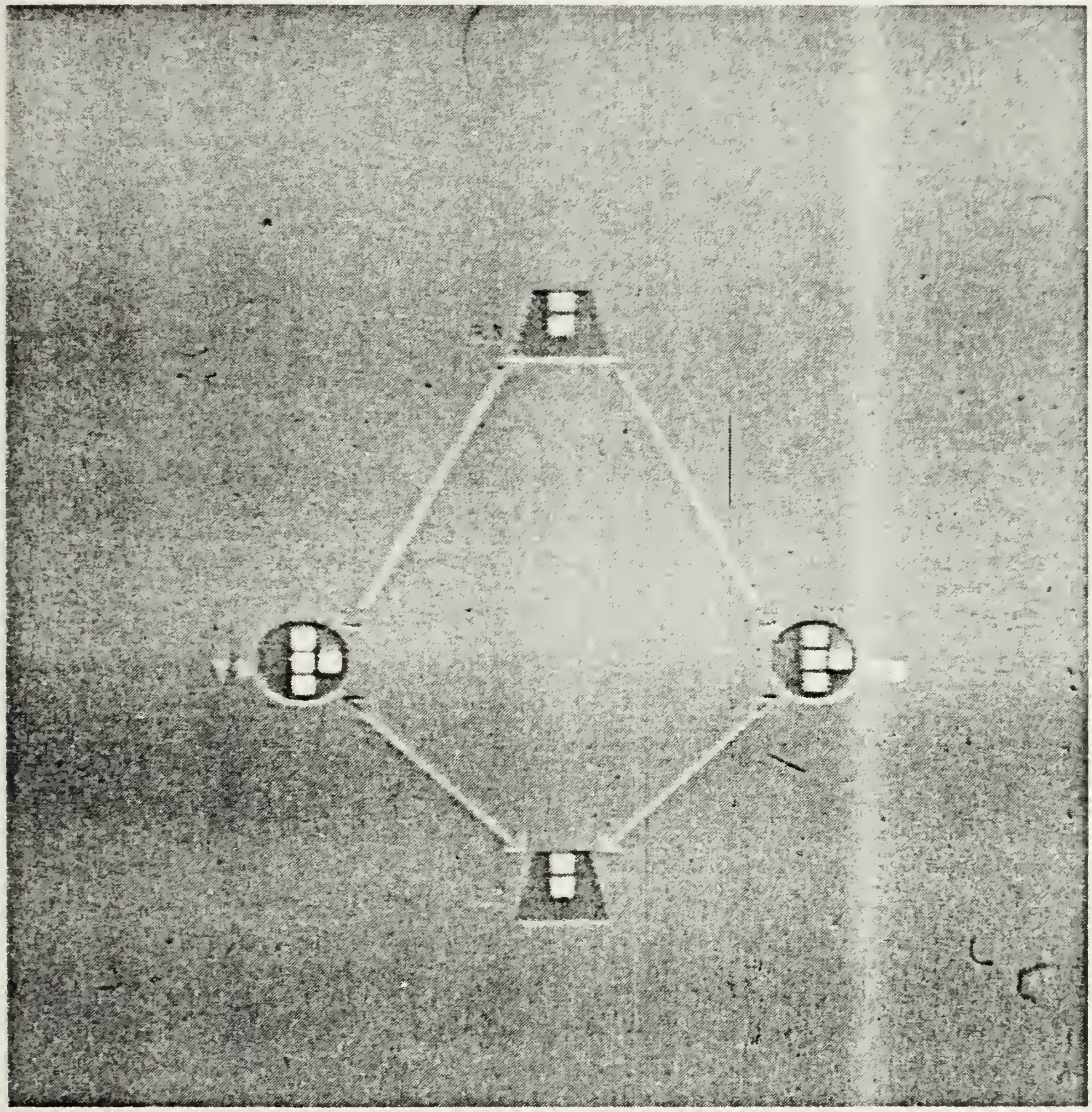

Figure 21.

RUN03   Page 1   Fri Feb 8 05:09:47 1980

```
10
T1 150 100 1
T2 150 250 1
T3 150 400 1
T4 350 250 1
G1 0 0 0
G2 0 0 0
G3 0 0 0
G4 0 0 0
G5 0 0 0
G6 0 0 0
8
TR1 250 175 1
2 1 2
1 4
TR2 250 325 1
2 2 3
2 4 8
TR3 0 0 0
1 5
2 1 6
TR4 0 0 0
1 6
1 10
TR5 0 0 0
1 7
2 2 9
TR6 0 0 0
1 8
1 3
TR7 0 0 0
1 9
1 7
TR8 0 0 0
1 10
1 5
MARK 5 1
MARK 7 1
MARK 3 1
END
```

```
NXTEVT= 11
                 1        0       150       100        1        2
                 3        0       150       250        1        2
                 5        1       150       400        1        2
                 7        0       350       250        1        2
                 9        1         0         0        0        2
                11        0         0         0        0        2
                13        1         0         0        0        2
                15        0         0         0        0        2
                17        0         0         0        0        2
                19        0         0         0        0        2

NXTTRN=    9
           21        1        4       250       175        1        3
           24        6        7       250       325        1        3
           27       12       14         0         0        0        3
           30       17       19         0         0        0        3
           33       21       23         0         0        0        3
           36       25       26         0         0        0        3
           39       30       32         0         0        0        3
           42       34       35         0         0        0        3

IOTABLE:
   2    1    2    1    4    2    2    3    2    4
   8    1    5    2    1    6    1    6    1   10
   1    7    2    2    9    1    8    1    3    1
   9    1    7    1   10    1    5    0

NAMES:    NXTVAR= 45
T1T2T3T4G1G2G3G4G5G6TR1TR2TR3TR4TR5TR6TR7TR8
```

RUN03B Page 1 Fri Feb 8 05:19:47 1980

1	1	1	1	0	0	1	0	0	1	0
2	0	0	1	1	0	0	1	0	0	1
3	0	1	1	1	1	0	0	0	1	0
4	1	0	0	2	0	1	1	1	0	0
5	1	1	1	2	0	0	0	0	1	1
6	0	0	1	3	1	0	1	0	0	0
7	1	1	1	3	0	1	0	0	1	0
8	0	0	1	4	0	0	1	0	0	1
9	0	1	1	4	1	0	0	0	1	0
10	1	0	0	5	0	1	1	1	0	0
11	1	1	1	5	0	0	0	0	1	1
12	0	0	1	6	1	0	1	0	0	0
13	1	1	1	6	0	1	0	0	1	0
14	0	0	1	7	0	0	1	0	0	1
15	0	1	1	7	1	0	0	0	1	0
16	1	0	0	8	0	1	1	1	0	0
17	1	1	1	8	0	0	0	0	1	1
18	0	0	1	9	1	0	1	0	0	0
19	1	1	1	9	0	1	0	0	1	0
20	0	0	1	10	0	0	1	0	0	1
21	0	1	1	10	1	0	0	0	1	0
22	1	0	0	11	0	1	1	1	0	0
23	1	1	1	11	0	0	0	0	1	1
24	0	0	1	12	1	0	1	0	0	0
25	1	1	1	12	0	1	0	0	1	0
26	0	0	1	13	0	0	1	0	0	1
27	0	1	1	13	1	0	0	0	1	0
28	1	0	0	14	0	1	1	1	0	0
29	1	1	1	14	0	0	0	0	1	1
30	0	0	1	15	1	0	1	0	0	0
31	1	1	1	15	0	1	0	0	1	0
32	0	0	1	16	0	0	1	0	0	1
33	0	1	1	16	1	0	0	0	1	0
34	1	0	0	17	0	1	1	1	0	0
35	1	1	1	17	0	0	0	0	1	1
36	0	0	1	18	1	0	1	0	0	0
37	1	1	1	18	0	1	0	0	1	0
38	0	0	1	19	0	0	1	0	0	1
39	0	1	1	19	1	0	0	0	1	0
40	1	0	0	20	0	1	1	1	0	0
41	1	1	1	20	0	0	0	0	1	1
42	0	0	1	21	1	0	1	0	0	0
43	1	1	1	21	0	1	0	0	1	0
44	0	0	1	22	0	0	1	0	0	1
45	0	1	1	22	1	0	0	0	1	0
46	1	0	0	23	0	1	1	1	0	0
47	1	1	1	23	0	0	0	0	1	1
48	0	0	1	24	1	0	1	0	0	0
49	1	1	1	24	0	1	0	0	1	0
50	0	0	1	25	0	0	1	0	0	1

```
 1 2
 2 3 5
 3 3
 4 1 4 7
 5 2
 6 5 8
 7 3
 8 2 3 7
 9 3
10 4 5 6
11 3
12 1 7 8
13 2
14 3 5
15 3
16 1 4 7
17 2
18 5 8
19 3
20 2 3 7
21 3
22 4 5 6
23 3
24 1 7 8
25 2
26 3 5
27 3
28 1 4 7
29 2
30 5 8
31 3
32 2 3 7
33 3
34 4 5 6
35 3
36 1 7 8
37 2
38 3 5
39 3
40 1 4 7
41 2
42 5 8
43 3
44 2 3 7
45 3
46 4 5 6
47 3
48 1 7 8
49 2
50 3 5
51 3
52 1 4 7
53 2
54 5 8
55 3
56 2 3 7
57 3
58 4 5 6
59 3
60 1 7 8
```

```
 61 2
 62 3 5
 63 3
 64 1 4 7
 65 2
 66 5 8
 67 3
 68 2 3 7
 69 3
 70 4 5 6
 71 3
 72 1 7 8
 73 2
 74 3 5
 75 3
 76 1 4 7
 77 2
 78 5 8
 79 3
 80 2 3 7
 81 3
 82 4 5 6
 83 3
 84 1 7 8
 85 2
 86 3 5
 87 3
 88 1 4 7
 89 2
 90 5 8
 91 3
 92 2 3 7
 93 3
 94 4 5 6
 95 3
 96 1 7 8
 97 2
 98 3 5
 99 3
100 1 4 7
```

```
122
S301212 250 100 1
S302212 250 100 1
S303112 250 100 1
S304112 250 100 1
S305012 250 100 1
S306012 250 100 1
S307312 250 100 1
S308312 250 100 1
S315312 250 100 1
S316312 250 100 1
S319312 250 100 1
S320312 250 100 1
R102205  75 200 1
R103105  75 200 1
P110105  75 200 1
R114205  75 200 1
R115305  75 200 1
R205005 425 200 1
R208305 425 200 1
R212005 425 200 1
R218205 425 200 1
R220305 425 200 1
T201216 250 300 1
T202216 250 300 1
T204116 250 300 1
T206016 250 300 1
T207316 250 300 1
T208316 250 300 1
T209116 250 300 1
T210116 250 300 1
T211016 250 300 1
T212016 250 300 1
T213216 250 300 1
T214216 250 300 1
T216316 250 300 1
T217216 250 300 1
T218216 250 300 1
T219316 250 300 1
T103108 100 400 1
T104108 100 400 1
T109108 100 400 1
T110108 100 400 1
T113208 100 400 1
T114208 100 400 1
T115308 100 400 1
T116308 100 400 1
T005008 400 400 1
T006008 400 400 1
T011008 400 400 1
T012008 400 400 1
T017208 400 400 1
T018208 400 400 1
T019308 400 400 1
T020308 400 400 1
B1      100 100 0
B2      100 100 0
B3      100 100 0
B4      100 100 0
B5      100 100 0
```

61	B6	100	100	0
62	B7	100	100	0
63	B8	100	100	0
64	B9	100	100	0
65	B10	100	100	0
66	B11	100	100	0
67	B12	100	100	0
68	B13	100	100	0
69	B14	100	100	0
70	B15	100	100	0
71	B16	100	100	0
72	B17	100	100	0
73	B18	100	100	0
74	B19	100	100	0
75	B20	100	100	0
76	B21	100	100	0
77	B22	100	100	0
78	B23	100	100	0
79	B24	100	100	0
80	B25	100	100	0
81	B26	100	100	0
82	B27	100	100	0
83	B28	100	100	0
84	B29	100	100	0
85	B30	100	100	0
86	B31	100	100	0
87	B32	100	100	0
88	B33	100	100	0
89	B34	100	100	0
90	A1	500	500	0
91	A2	500	500	0
92	A3	500	500	0
93	A4	500	500	0
94	A5	500	500	0
95	A6	500	500	0
96	A7	500	500	0
97	A8	500	500	0
98	A9	500	500	0
99	A10	500	500	0
100	A11	500	500	0.
101	A12	500	500	0
102	A13	500	500	0
103	A14	500	500	0
104	A15	500	500	0
105	A16	500	500	0
106	A17	500	500	0
107	A18	500	500	0
108	A19	500	500	0
109	A20	500	500	0
110	A21	500	500	0
111	A22	500	500	0
112	A23	500	500	0
113	A24	500	500	0
114	A25	500	500	0
115	A26	500	500	0
116	A27	500	500	0
117	A28	500	500	0
118	A29	500	500	0
119	A30	500	500	0
120	A31	500	500	0

```
A32      500 500 0
A33      500 500 0
A34      500 500 0
68
TR1      220 200 1
2 1 56
1 23
TR2      140 140 1
2 2 58
1 13
TR3      140 260 1
2 13 60
1 24
TR4      100 100 0
2 3 62
1 14
TR5      100 100 0
2 14 64
1 39
TR6      100 100 0
2 4 66
1 25
TR7      100 100 0
2 25 68
1 40
TR8      330 160 1
2 5 70
1 18
TR9      375 320 1
2 18 72
1 47
TR10     100 100 0
2 6 74
1 26
TR11     100 100 0
2 26 76
1 48
TR12     280 200 1
2 27 78
1 7
TR13     365 260 1
2 28 80
1 19
TR14     100 100 0
2 19 82
1 8
TR15     175 400 1
2 29 84
1 41
TR16     170 230 1
2 30 86
1 15
TR17      50 300 1
2 15 88
1 42
TR18     450 300 1
2 54 92
1 22
TR19     360 140 1
2 22 90
```

```
181 1 12
182 TR20      100 100 0
183 2 53 96
184 1 38
185 TR21      100 100 0
186 2 38 94
187 1 11
188 TR22      100 100 0
189 2 52 100
190 1 21
191 TR23      330 230 1
192 2 21 98
193 1 37
194 TR24      325 335 1
195 2 51 102
196 1 36
197 TR25      100 100 0
198 2 46 106
199 1 35
200 TR26      100 100 0
201 2 35 104
202 1 10
203 TR27      100 350 1
204 2 45 108
205 1 17
206 TR28      170 160 1
207 2 17 110
208 1 9
209 TR29      100 100 0
210 2 44 112
211 1 16
212 TR30      100 100 0
213 2 16 114
214 1 34
215 TR31      180 310 1
216 2 43 116
217 1 33
218 TR32      100 100 0
219 2 32 120
220 1 20
221 TR33      100 100 0
222 2 20 118
223 1 50
224 TR34      300 400 1
225 2 31 122
226 1 49
227 TR35      100 100 0
228 1 55
229 2 57 56
230 TR36      100 100 0
231 1 57
232 2 59 58
233 TR37      100 100 0
234 1 59
235 2 60 61
236 TR38      100 100 0
237 1 61
238 2 63 62
239 TR39      100 100 0
240 1 63
```

```
241 2 65 64
242 TR40     100 100 0
243 1 65
244 2 67 66
245 TR41     100 100 0
246 1 67
247 2 69 68
248 TR42     100 100 0
249 1 69
250 2 71 70
251 TR43     100 100 0
252 1 71
253 2 73 72
254 TR44     100 100 0
255 1 73
256 2 75 74
257 TR45     100 100 0
258 1 75
259 2 77 76
260 TR46     100 100 0
261 1 77
262 2 79 78
263 TR47     100 100 0
264 1 79
265 2 81 80
266 TR48     100 100 0
267 1 81
268 2 83 82
269 TR49     100 100 0
270 1 83
271 2 85 84
272 TR50     100 100 0
273 1 85
274 2 87 86
275 TR51     100 100 0
276 1 87
277 2 89 88
278 TR52     100 100 0
279 1 89
280 2 91 92
281 TR53     100 100 0
282 1 91
283 2 93 90
284 TR54     100 100 0
285 1 93
286 2 95 96
287 TR55     100 100 0
288 1 95
289 2 97 94
290 TR56     100 100 0
291 1 97
292 2 99 100
293 TR57     100 100 0
294 1 99
295 2 101 98
296 TR58     100 100 0
297 1 101
298 2 103 102
299 TR59     100 100 0
300 1 103
```

```
301 2 105 106
302 TR60     100 100 0
303 1 105
304 2 107 104
305 TR61     100 100 0
306 1 107
307 2 109 108
308 TR62     100 100 0
309 1 109
310 2 111 110
311 TR63     100 100 0
312 1 111
313 2 113 112
314 TR64     100 100 0
315 1 113
316 2 115 114
317 TR65     100 100 0
318 1 115
319 2 117 116
320 TR66     100 100 0
321 1 117
322 2 119 120
323 TR67     100 100 0
324 1 119
325 2 121 118
326 TR68     100 100 0
327 1 121
328 2 55 122
329 MARK 1 1
330 MARK 2 1
331 MARK 3 1
332 MARK 4 1
333 MARK 5 1
334 MARK 6 1
335 MARK 27 1
336 MARK 28 1
337 MARK 29 1
338 MARK 30 1
339 MARK 31 1
340 MARK 32 1
341 MARK 43 1
342 MARK 44 1
343 MARK 45 1
344 MARK 46 1
345 MARK 51 1
346 MARK 52 1
347 MARK 53 1
348 MARK 54 1
349 MARK 61 1
350 END
```

THESIS-RELATED PROGRAMS & TEXT

FILE	DESCRIPTION	VERSION	EXECUTION
DIRECTORY	LIST OF THESIS FILES	-	aa DIRECTORY
RUN01X	GENERATOR	2	transgraph
RUN02X	ENABLE TRANSITION	2	transgraph
RUN03X	DYNAMIC CONFLICTS	2	transgraph
RUN04X	HIERARCHICAL STRUCTURE	2	linkgraph
RUN05X	HIERARCHICAL STRUCTURE	2	transgraph
RUN06X	RING STRUCTURE	2	linkgraph
RUN07X	FEEDBACK GENERATOR	1	transgraph
RUN09X	DEPICTS SYNCHRONIZATION	1	transgraph
RUN11X	FOR MULTIPLE INPUTS (5)	2	transgraph
RUN12X	ALL-[illegible] INPUT	2	transgraph
RUN13X	CONSUMER-PRODUCER	2	transgraph
RUN14X	CONSUMER-PRODUCER	2	transgraph
RUN15X	TRIAL RUN RUN09X	2	linkgraph
RUN16X	SYNCHRONIZATION II	2	transgraph
RUN20X	PACKET-RADIO NETWORK	3	linkgraph
RUN21X	SIMPLE MULTI-ROUTING	3	link/transgraph
RUN22X	CONCURRENT NETWORK	3	linkgraph
RUN23X	TIME-SLOTTED COMMUNICATIONS	3	linkgraph
RUN24X	PRIORITY TIME-SLOTTED	3	linkgraph
RUN25X	CONCURRENT NETWORK (-)	3	linkgraph
RUN26X	TIME-SLOTTED COMMUNICATION(-)	3	linkgraph
RUN27X	PRIORITY TIME-SLOTTED (-)	3	linkgraph
RUN01-29	INPUT FILES TO SIMULATOR	-	simulator.out
advisors.c	THESIS ADVISORS	-	advisors
end.c	END FRAME FOR FILM	-	end
experiment.c	THROUGHPUT RESULTS OF EXPRMNT	-	experiment
intro.c	INTRODUCTION FOR FILM	-	intro
linear.c	NODE-FILE SIZE RELATIONSHIP	-	linear
linkgraph.c	[illegible]-LINK GRAPHICS	-	linkgraph
nps.c	TITLE FRAME FOR NPS	-	nps
outline.c	SOFTWARE MODULES	-	outline
simulator	PETRI-NET SIMULATOR	-	simulator.out
title.c	TITLE FRAME FOR FILM	-	title
transgraph.c	TRANSITION GRAPHICS	-	transgraph

```
      PROGRAM SIMULATOR

C     ORIGINAL VERSION OF THIS PROGRAM (PROGRAM TESTNT) WRITTEN BY L.A.COX□
C     MODIFIED TO OPERATE ON UNIX (PDP 11/70) BY S.C.JENNINGS & R.J.HARTEL□
C     PROGRAM SIMULATOR READS USER INPUT FILE AND PRODUCES 6 OUTPUT FILES

C     MAINLINE

      CALL INIT
      CALL INPUT1
      CALL DUMPPP
      CALL MOVENET(150)
      CALL EXIT
      END


      SUBROUTINE INIT

C     INIT OPENS USER INPUT FILE & CREATES 6 OUTPUT FILES
C     FNAM1 STORES THE INPUT FILE -------------------------- RUN..
C     FNAM2 STORES THE FORMATTED INPUT DATA STRUCTURES ----- RUN..A
C     FNAM3 STORES THE FORMATTED ITERATIONS OF THE NETWORK - RUN..B
C     FNAM4 STORES THE FORMATTED LINKS OF THE NETWORK ------ RUN..C
C     FNAM5 STORES THE UNFORMATTED GRAPHICS INPUT ---------- RUN..X
C     FNAM6 STORES THE UNFORMATTED GRAPHICS ITERATIONS ----- RUN..Y
C     FNAM7 STORES THE UNFORMATTED GRAPHICS STATES --------- RUN..Z


      BYTE      FNAM1
      BYTE      FNAM2
      BYTE      FNAM3
      BYTE      FNAM4
      BYTE      FNAM5
      BYTE      FNAM6
      BYTE      FNAM7

      COMMON/USRFILE/FNAM1(6),FNAM2(7),FNAM3(7),FNAM4(7),FNAM5(7),FNAM6(7)□
     1                FNAM7(7)
      COMMON/EVENT/IEVENT(400,6),NXTEVT
      COMMON/TRANS/ITRANS(400,7),NXTTRN,IINTR,IISTORE(100)
      BYTE NAMES
      COMMON/NAME/NAMES(4000),NXTNAM
      COMMON/IOTAB/IOTABL(4000),NXTTRE
 1000 FORMAT(' INITIALIZING PROGRAM')
      TYPE 1000
 1002 FORMAT(' BEGIN TEST-GRAPH-NET')
      TYPE 1002
 2007 TYPE 2000
 2000 FORMAT  (' ***-> INPUT FILE?   NAME MUST BE ENTERED AS: RUN01 - RUN9□
      ACCEPT 2001, FNAM1
 2001 FORMAT (6A1)
      FNAM1(6) = 0
      OPEN (UNIT = 1, NAME = FNAM1, TYPE = 'OLD', ERR = 2006)
      GO TO 2004
 2006  TYPE 2002, FNAM1
 2002 FORMAT (' ERROR OPENING FILE ',X6A1)
      GO TO 2007
 2004 DO 2005  I = 1, 5
      FNAM2(I) = FNAM1(I)
      FNAM3(I) = FNAM1(I)
```

```
      FNAM4(I) = FNAM1(I)
      FNAM5(I) = FNAM1(I)
      FNAM6(I) = FNAM1(I)
      FNAM7(I) = FNAM1(I)
 2005 CONTINUE
      FNAM2(6) = 'A'
      FNAM2(7) = 0
      FNAM3(6) = 'B'
      FNAM3(7) = 0
      FNAM4(6) = 'C'
      FNAM4(7) = 0
      FNAM5(6) = 'X'
      FNAM5(7) = 0
      FNAM6(6) = 'Y'
      FNAM6(7) = 0
      FNAM7(6) = 'Z'
      FNAM7(7) = 0
      NXTEVT=1
      NXTTRN=1
      NXTNAM=1
      NXTTRE=1
 3000 FORMAT(' INITIALIZATION COMPLETE')
      TYPE 3000
      RETURN
      END


      SUBROUTINE DUMPPP

C     OPENS FNAM2 & FNAM4
      BYTE      FNAM1
      BYTE      FNAM2
      BYTE      FNAM3
      BYTE      FNAM4
      BYTE      FNAM5
      BYTE      FNAM6
      BYTE      FNAM7
      COMMON/USRFILE/FNAM1(6),FNAM2(7),FNAM3(7),FNAM4(7),FNAM5(7),FNAM6(7),
     1           FNAM7(7)
      COMMON/EVENT/IEVENT(400,6),NXTEVT
      COMMON/TRANS/ITRANS(400,7),NXTTRN,IINTR,IISTORE(100)
      COMMON/IOTAB/IOTABL(4000),NXTTRE
      BYTE NAMES
      COMMON/NAME/NAMES(4000),NXTNAM
      OPEN(UNIT=1,NAME=FNAM2,TYPE='NEW',INITIALSIZE=40000)
 1000 FORMAT('  NXTEVT=',I4)
 1001 FORMAT(5X,6I8)
      WRITE(1,1000) NXTEVT
      DO 1500 I=1,NXTEVT-1
 1500 WRITE(1,1001) (IEVENT(I,J),J=1,6)
 2000 FORMAT(/,'  NXTTRN=',I4)
 2001 FORMAT(1X,7I8)
      WRITE(1,2000) NXTTRN
      DO 2500 I=1,NXTTRN-1
      WRITE(1,2001) (ITRANS(I,J),J=1,7)
 2500 CONTINUE

 3000 FORMAT(/,' IOTABLE:',/,60(10I4,/))
      WRITE(1,3000) (IOTABL(I),I=1,NXTTRE)
 4000 FORMAT(/,'  NAMES:   NXTNAM=',I4)
```

```
 4001 FORMAT(X,100A1)
      WRITE(1,4000) NXTNAM
      WRITE(1,4001) (NAMES(I),I=1,NXTNAM)
 5000 FORMAT(1H1)
      WRITE(1,5000)
      CLOSE(UNIT=1,DISPOSE='SAVE')
      OPEN(UNIT=1,NAME=FNAMS,TYPE='NEW',INITIALSIZE=14000,
     1  FORM='UNFORMATTED')
      WRITE(1) (NXTEVT)
      DO 5001 I=1,NXTEVT-1
 5001 WRITE(1) (IEVENT(I,J),J=1,6)
      WRITE(1) (NXTTRN)
      DO 5002 I=1,NXTTRN-1
 5002 WRITE(1) (ITRANS(I,J),J=1,7)
      WRITE(1) (NXTTRE)
      WRITE(1) (IOTABL(I),I=1,NXTTRE)
      WRITE(1) (NXTNAM)
      WRITE(1) (NAMES(I),I=1,NXTNAM)
      CLOSE(UNIT=1,DISPOSE='SAVE')
      RETURN
      END
      SUBROUTINE INPUT1
      COMMON/SCAN/IWORD(15,10),NUMBER
 8000 FORMAT('      INPUT BEGINS')
 8001 FORMAT('      INPUT COMPLETE')
      TYPE 8000
      I=0
      CALL SCANR
      CALL XINTGR(1,I)
      DO 1000 J=1,I
      CALL INPUTE
 1000 CONTINUE
      CALL INPUTT
 2000 CONTINUE
      CALL SCANR
      IF(MATCHS(1,'END',3).EQ.1) GO TO 3000
      IF(MATCHS(1,'MARK',4).EQ.1) CALL MARKER
      GO TO 2000
 3000 TYPE 8001
      CLOSE(UNIT=1,DISPOSE='SAVE')
      RETURN
      END
      SUBROUTINE INPUTE
      COMMON/EVENT/IEVENT(400,6),NXTEVT
      BYTE IWORD
      COMMON/SCAN/IWORD(15,10),NUMB
      J=0
C     READ A SINGLE EVENT LINE FROM INPUT AND
C     STORE IT APPROPRIATELY
      CALL SCANR
```

```
      CALL STONAM(1,IEVENT(NXTEVT,1),IEVENT(NXTEVT,6))

      DO 1000 I=2,4
      CALL XINTGR(I,J)
 1000 IEVENT(NXTEVT,I+1)=J

      NXTEVT=NXTEVT+1
      IF(NXTEVT.GT.400) GO TO 9000
      RETURN
 9000 TYPE 9900
 9900 FORMAT(' EVENT/PLACE TABLE OVERFLOW')
      CALL EXIT
      RETURN
      END


      SUBROUTINE INPUTT

      COMMON/TRANS/ITRANS(400,7),NXTTRN,IINTR,IISTORE(100)
      BYTE IWORD
      COMMON/SCAN/IWORD(15,10),NUMB

      I=0
      K=0
      CALL SCANR
      CALL XINTGR(1,I)

      DO 1000 J=1,I
      CALL SCANR
      CALL STONAM(1,ITRANS(NXTTRN,1),ITRANS(NXTTRN,7))

     DO 2000 L=1,3
     LL=L+3
     CALL XINTGR(L+1,K)
 2000 ITRANS(NXTTRN,LL)=K

      CALL SCANR
      CALL STOIOT(NUMB,ITRANS(NXTTRN,2))
      CALL SCANR
      CALL STOIOT(NUMB,ITRANS(NXTTRN,3))
      NXTTRN=NXTTRN+1
      IF(NXTTRN.GT.400) GO TO 9000
 1000 CONTINUE

      RETURN

 9000 TYPE 9900
 9900 FORMAT(' TRANSITION TABLE OVERFLOW')
      CALL EXIT
      RETURN
      END


      SUBROUTINE STONAM(NWORD,NPOINT,KOUNT)

C           STORE STRING 'NWORD' FROM SCANNER INTO
C           NAME TABLE AND RETURN A POINTER 'NPOINT'

      BYTE IWORD,NAMES,BLANK
      COMMON/SCAN/IWORD(15,10),NUMB
```

```
      COMMON/NAME/NAMES(4000),NXTNAM
      DATA BLANK/1H /
      DO 1000 I=1,10
      KOUNT=I-1
      IF(IWORD(NWORD,I).EQ.BLANK) GO TO 2000
 1000 CONTINUE
 2000 CONTINUE

      IF(NXTNAM+KOUNT .GT.4000) GO TO 9000

      DO 3000 I=1,KOUNT
 3000 NAMES(NXTNAM+I-1)=IWORD(NWORD,I)

      NPOINT=NXTNAM
      NXTNAM=NXTNAM+KOUNT
      RETURN

 9000 TYPE 9900
 9900 FORMAT(' NAME TABLE OVERFLOW')
      CALL EXIT
      RETURN
      END


      SUBROUTINE STOIOT(NUMBER,LINK)

      COMMON/IOTAB/IOTABL(4000),NXTTRE

C     STORE INPUTS AND OUTPUTS OF TRANSITIONS
C     IN THE TABLE, RETURN THE LINK

      IF(NXTTRE+NUMBER .GT. 4000) GO TO 9000

      K=0
      DO 1000 I=1,NUMBER
      CALL XINTGR(I,K)
      J=(NXTTRE-1)+I
 1000 IOTABL(J)=K

      LINK=NXTTRE
      NXTTRE=NXTTRE+NUMBER
      RETURN

 9000 TYPE 9900
 9900 FORMAT(' IO TABLE OVERFLOW (TRANSITIONS)')
      CALL EXIT
      RETURN
      END


      SUBROUTINE MARKER

      COMMON/EVENT/IEVENT(400,6),NXTEVT
      I=0
      J=0
      CALL XINTGR(2,I)
      CALL XINTGR(3,J)
      IF(I.LT.1 .OR. I.GT.NXTEVT) RETURN
      IEVENT(I,2)=J
      RETURN
```

```
      END



      SUBROUTINE MOVENET(NTIMES)

C     EXECUTE THE PETRI-NET 'NTIMES' OR STEPS

      COMMON/TRANS/ITRANS(400,7),NXTTRN,IINTR,IISTORE(100)
      COMMON/EVENT/IEVENT(400,6),NXTEVT
      BYTE        FNAM1
      BYTE        FNAM2
      BYTE        FNAM3
      BYTE        FNAM4
      BYTE        FNAM5
      BYTE        FNAM6
      BYTE        FNAM7
      COMMON/USRFILE/FNAM1(6),FNAM2(7),FNAM3(7),FNAM4(7),FNAM5(7),FNAM6(7),
     1                FNAM7(7)
      DATA ITIME/1/

 1000 FORMAT(' EXECUTING  TIME=',I4)

      OPEN(UNIT=1,NAME=FNAM3,TYPE='NEW',INITIALSIZE=120000)
      OPEN(UNIT=2,NAME=FNAM6,TYPE='NEW',FORM='UNFORMATTED',
     1  INITIALSIZE=120000)
      OPEN(UNIT=3,NAME=FNAM4,TYPE='NEW',INITIALSIZE=12000)
      OPEN(UNIT=4,NAME=FNAM7,TYPE='NEW',FORM='UNFORMATTED',
     1  INITIALSIZE=12000)


      DO 2000 I=1,NTIMES
      TYPE 1000,ITIME
      IINTR=0
      CALL MOVE
  106 FORMAT(35I2)
      WRITE(1,106) (IEVENT(J,2),J=1,NXTEVT-1)
      WRITE(2) (IEVENT(J,2),J=1,NXTEVT-1)
  107 FORMAT(I3)
      WRITE(3,107)  IINTR
  108 FORMAT(100I3)
      WRITE(3,108)  (IISTORE(J), J=1,IINTR)
      WRITE(4) IINTR
      WRITE(4) (IISTORE(J),J=1,IINTR)

      ITIME=ITIME+1
 2000 CONTINUE

      CLOSE(UNIT=4,DISPOSE='SAVE')
      CLOSE(UNIT=3,DISPOSE='SAVE')
      CLOSE(UNIT=2,DISPOSE='SAVE')
      CLOSE(UNIT=1,DISPOSE='SAVE')

      RETURN
      END


      SUBROUTINE MOVE

C     EXECUTE THE NET ONE STEP
```

```

      COMMON/TRANS/ITRANS(400,7),NXTTRN,IINTR,IISTORE(100)
      DIMENSION MARKS(400)
      ITEST=0

C     CHECK ALL TRANSITIONS TO SEE WHICH ARE ENABLED
      DO 0500 I=1,NXTTRN-1
 0500 MARKS(I)=NABLED(I)
      DO 1000 I=1,NXTTRN-1
      IF(MARKS(I).EQ.0) GO TO 1000
 0600 FORMAT(' DYNAMIC CONFLICT, TR#=',I4)

      IF(NABLED(I).EQ.1) GO TO 0800
      TYPE 0600,I
      GO TO 1000
 0800 CONTINUE
      CALL UNMARK(I,ITEST)
      IF(ITEST.EQ.1) GO TO 1000
      IINTR=IINTR+1
      IISTORE(IINTR)=I
      CALL MARKEM(I)
 1000 CONTINUE

      RETURN
      END


      FUNCTION NABLED(NUMBER)

C     RETURN 1 IF TRANSITION # 'NUMBER' IS ENABLED, READY
C     TO FIRE.  ELSE RETURN 0.

      COMMON/TRANS/ITRANS(400,7),NXTTRN
      COMMON/EVENT/IEVENT(400,6),NXTEVT
      COMMON/IOTAB/IOTABL(4000),NXTTRE

C     CHECK LIST OF INPUTS TO SEE IF ALL ARE MARKED

      MARK=0
      IPT=ITRANS(NUMBER,2)
      KOUNT=IOTABL(IPT)

      DO 1000 I=IPT+1,IPT+KOUNT
      NEVENT=IOTABL(I)
      IF(IEVENT(NEVENT,2).GT.0) MARK=MARK+1
 1000 CONTINUE

      NABLED=0
      IF(KOUNT.EQ.MARK) NABLED=1
      RETURN
      END


      SUBROUTINE UNMARK(NUMBER,IERROR)
C
C           UNMARK (IE. DECREMENT THE NUMBER OF TOKENS
C                 ALL OF THE INPUT EVENTS TO TRANSITION # 'NUMBER'.
C                 RETURN IERROR=0 ......UNLESS......
C                 ONE EVENT IS A MULTIPLE INPUT OF THE SAME
C                 TRANSITION AND WE DONT HAVE ENOUGH MARKERS.
```

```
C                WHEN THIS HAPPENS, REPLACE ANY REMOVED
C                TOKENS AND RETURN IERROR=1.
C
      COMMON/EVENT/IEVENT(400,6),NXTEVT
      COMMON/TRANS/ITRANS(400,7),NXTTRN
      COMMON/IOTAB/IOTABL(4000),NXTTRE

      IPT=ITRANS(NUMBER,2)
      KOUNT=IOTABL(IPT)
      IERROR=0

      DO 1000 I=IPT+1,IPT+KOUNT
      NEVENT=IOTABL(I)
      J=I
      IEVENT(NEVENT,2)=IEVENT(NEVENT,2)-1
      IF(IEVENT(NEVENT,2).LT.0) GO TO 2000
 1000 CONTINUE
      RETURN

 2000 CONTINUE
      DO 3000 I=IPT+1,J
      NEVENT=IOTABL(I)
 3000 IEVENT(NEVENT,2)=IEVENT(NEVENT,2)+1
      IERROR=1

      RETURN
      END


      SUBROUTINE MARKEM(NUMBER)
C
C          MARK ALL OUTPUT EVENTS OF TRANSITION # 'NUMBER'
C
      COMMON/EVENT/IEVENT(400,6),NXTEVT
      COMMON/TRANS/ITRANS(400,7),NXTTRN
      COMMON/IOTAB/IOTABL(4000),NXTTRE

      IPT=ITRANS(NUMBER,3)
      KOUNT=IOTABL(IPT)
      DO 1000 I=IPT+1,IPT+KOUNT
      NEVENT=IOTABL(I)
 1000 IEVENT(NEVENT,2)=IEVENT(NEVENT,2)+1
      RETURN
      END

      SUBROUTINE SCANR

      BYTE IWORD,ISC,IBLANK
      COMMON/SCAN/IWORD(15,10),NUMBER
      BYTE NBUFFR
      COMMON/SCAN1/NBUFFR(80)
      DATA ISC/1H;/
      DATA IBLANK/1H /
 0001 FORMAT(80A1)
      READ(1,0001,END=9999,ERR=9999) (NBUFFR(I),I=1,80)
      IPOINT=1
C          SET POINTER TO FIRST CHARACTER IN THE BUFFER
C          NOW PROCESS THE FIRST 15 TOKENS DELIMITED BY EITHER
C          A BLANK (OR MULTIPLE BLANKS) OR A SEMICOLON.
C
```

```
      DO 0002 NUMBER=1,15
      IFLAG=0
C     SET IWORD(NUMBER,X)=IBLANK (SET WORD TO ALL BLANKS)
      DO 0003 I=1,10
 0003 IWORD(NUMBER,I)=IBLANK
C     START SCANNING LINE FROM POINTER ON TO FIND NON-BLANK
      KOUNT=1
C     "KOUNT" KEEPS TRACK OF THE NO. OF CHAR. IN THE TOKEN
      DO 0004 KPOINT=IPOINT,80
      IF(NBUFFR(KPOINT).NE.IBLANK .AND. NBUFFR(KPOINT).NE.ISC)
     1          GO TO 0005
      IF(IFLAG.EQ.0) GO TO 0004
      IF(IFLAG.EQ.1) GO TO 0006
 0005 CONTINUE
      IFLAG=1
      IWORD(NUMBER,KOUNT)=NBUFFR(KPOINT)
      KOUNT=KOUNT+1
      IF(KOUNT.GT.10) GO TO 0006
 0004 CONTINUE

 0006 CONTINUE
C     END OF TOKEN FOUND, RESET SOME POINTERS
      IPOINT=KPOINT+1
      IF(IPOINT.GT.80) GO TO 0010

 0002 CONTINUE
C     END OF BASIC TOKEN GETTING LOOP

 0010 NUMBER=NUMBER-1
      RETURN
 9999 CONTINUE
C     END OF FILE OR I/O ERROR DETECTED
 9998 FORMAT(' EOF OR ERROR ON SCANNER INPUT FROM UNIT 1')
      TYPE 9998
      NUMBER=0
      RETURN
      END



      SUBROUTINE XINTGR(NWORD,IVALUE)

C         CONVERT THE ENTRY IN "IWORD" TO INTEGER
C            RETURN INTEGER "IVALUE"

      BYTE IWORD
      COMMON/SCAN/IWORD(15,10),NUMBER
      BYTE TSTRNG
      DIMENSION TSTRNG(10)
      BYTE IBLANK
      DATA IBLANK/1H /
      DO 0001 I=1,10
      KOUNT=I
      TSTRNG(I)=IWORD(NWORD,I)
      IF(IWORD(NWORD,I).EQ.IBLANK) GO TO 1000
 0001 CONTINUE
 1000 CONTINUE
      KOUNT=KOUNT-1

 2004 FORMAT(X,10A1)
```

```
 2005 FORMAT(X,1I10)
      DO 2006 I=1,KOUNT
      J=11-I
      K=(KOUNT+1)-I
 2006 TSTRNG(J)=TSTRNG(K)
      L=10-KOUNT
      DO 2007 I=1,L
 2007 TSTRNG(I)=IBLANK
      OPEN(UNIT=2,NAME='ISTORE',TYPE='NEW',INITIALSIZE=20)
      WRITE(2,2004) (TSTRNG(I),I=1,10)
      CLOSE(UNIT=2,DISPOSE='SAVE')
      OPEN(UNIT=2,NAME='ISTORE',TYPE='OLD')
      READ(2,2005) IVALUE
      CLOSE(UNIT=2,DISPOSE='DELETE')
      RETURN
      END


      FUNCTION MATCHS(NUMB,STRING,NCHAR)

C          THIS FUNCTION DETERMINES IF SCANNER TOKEN
C          IWORD(NUMB) MATCHES THE CHARACTERS IN "STRING"
C          AT LEAST FOR THE FIRST "NCHAR" CHARACTERS.

C          IF THERE IS A MATCH, IT RETURNES THE INTEGER "1"
C          NO MATCH RETURNS "0".

      BYTE IWORD
      COMMON/SCAN/IWORD(15,10),NUMBER
      BYTE STRING
      DIMENSION STRING(10)
      MATCHS=0

      DO 0001 I=1,NCHAR
      IF(IWORD(NUMB,I).NE.STRING(I)) RETURN
 0001 CONTINUE

C      IF YOU GET HERE, THEY WERE THE SAME...
      MATCHS=1
      RETURN
      END


C     END OF PROGRAM SIMULATOR
```

```
#

       /*************************************************************/
       /*************************************************************/
       /********                                            ********/
       /********            PROGRAM TRANSGRAPH.C            ********/
       /********                                            ********/
       /********       STEPHEN C. JENNINGS JC91 USMC        ********/
       /********        ROBERT J. HARTEL CS91 USA           ********/
       /********                                            ********/
       /********         WRITTEN FALL QUARTER 1979          ********/
       /********         NAVAL POSTGRADUATE SCHOOL          ********/
       /********           MONTEREY, CALIFORNIA             ********/
       /********                                            ********/
       /*************************************************************/
       /*************************************************************/


/*********************************/
/**   EXTERNAL DECLARATIONS   **/
/*********************************/


/*** LITERALS ***/

#define  header      4
#define  pictures    50
#define  bounds      100
#define  limit       500


/*** STRUCTURES ***/

struct {                         /* data structure information on net nodes .. */

    int ctrl1;             /* store control char not used in program ... */
    int nameptr;           /* index to names array ..................... */
    int marker;            /* initial marker state of the network ...... */
    int xcord;             /* x cordinate of place ..................... */
    int ycord;             /* y cordinate of place ..................... */
    int plot;              /* whether or not place is to be plotted .... */
    int length;            /* length of name associated with place ..... */

}file1 [bounds],  *bp1;     /* pointer into data structure .............. */


struct {                         /* data structure information on transitions. */

    int ctrl2;             /* store control char not used in program ... */
    int trnptr;            /* index to names array ..................... */
    int intrn;             /* pointer to inputs for a transition ....... */
    int outtrn;            /* pointer to outputs for a transition ...... */
    int xxcord;            /* x cordinate of transition ................ */
    int yycord;            /* y cordinate of transition ................ */
    int trnplot;           /* whether or not transition is to be plotted */
    int trnlen;            /* length of name associated with transition  */

}file2 [bounds],  *bp2;     /* pointer into data structure .............. */
```

```
/*** INTEGERS ***/

int a1,a2,a3,a4;              /* global storage for each data structure ... */
int buffer[bounds];           /* buffer into which each frame is read ..... */
int cntrl[1];                 /* variable containing # transitions fired .. */
int ctroverflow;              /* keep track of nodes overflow status ...... */
int dfltcolor;                /* a default color for indicating overflows . */
int fdfbuf;                   /* file descriptor for RUN..Y files ......... */
int fdgbuf;                   /* file descriptor for RUN..Z files ......... */
int firing[bounds];           /* storage into which fired places are read . */
int ictr;                     /* counter passed to a function ............. */
int ievents;                  /* number of non- & displayable nodes ....... */
int iflag;                    /* counter for the interrupt mechanism ...... */
int iotbl[limit];             /* forms input-to-output relationship ....... */
int kpictures;                /* counter for the iterations of the network. */
int linktbl[100][4];          /* version 1 & 2 screen nodes locations ..... */
int nbrplot;                  /* counter for number of displayed nodes .... */
int nbytes[2];                /* store count fields for data structures ... */
int overflowtbl[100][2];      /* data structure to store overflow locations */
int set;                      /* user selected conrac graphics screen ..... */
int sflag[20];                /* saves flag for later use by trnlite() .... */
int tblctr;                   /* a counter for version 3..reset conditions. */
int uniquetbl[100][4];        /* reset table locations for version 3 ...... */
int vers;                     /* user selected option ..................... */


/*** CHARACTERS ***/

char fbuf[20];                /* buffer to store name of second file ...... */
char gbuf[20];                /* buffer to store name of third file........ */
char names[limit];            /* character array for node labels .......... */
char scrn;                    /* option variable for display to the screen. */


/********************************/
/**       FUNCTION MAIN        **/
/********************************/

main() {

    extern rubout();     /* declare 'rubout' globally ................ */
    init();              /* read input file .......................... */
    determine();         /* verify if user wants to see data structure */
    display();           /* display input to crt ..................... */
    select();            /* select version of simulation & genisco set */
    prepare(2);          /* prepare genesco-conrac ................... */
    drawnode();          /* draw network nodes on conrac ............. */
    places();            /* verify correct nodes drawn ............... */
    trnslink();          /* function displays network transitions .... */
    imark();             /* starting status of network packets ....... */
    signal(2,rubout);    /* sets 'BRK' as interrupt .................. */
    marking();           /* successive iterations of network flow .... */
    gnfini();            /* closing out graphics facilities .......... */

}


/********************************/
/**   PROGRAM FUNCTIONS        **/
/********************************/
```

```


pause(peroid) {
     /* function necessary as sleep() not compatible with signal() */
     int i,j,k;

     printf("***>>interrupt......");
     for(i=0;i<peroid;i++) {
          for(j=0;j<400;j++) {
               for(k=0;k<1000;k++) {
               }
          }
     }
     printf("***>>wait......");
return;
}


rubout()  {
     /* function enables the 'brk' key as the interrupt signal */
     char halt;

     space(2);
     printf
     ("***>>> received signal...frame number %d...<ret> to continue \n",
     (iflag+2));
     printf
     ("***>>> for termination of program...type 'brk' from console \n");
     while ((halt=getchar())!='\n')  {
            /*  do-nothing loop  */
     }
     signal(2,rubout);
return;
}


space(returns) {
      int i;

      for (i=0; i<returns; i++) {
           printf("\n");
           }
return;
}


init() {
     /* function opens unformatted file & initializes start condition */
     int a,bufctr,count,fd,i,j;
     char cbuf[20],c,f;

     space(2);
     printf("***->TRANSGRAPH ILLUSTRATES PETRINET SIMULATION MODELS");
     space(2);
     printf("***->ENTER THE NAME OF THE FILE TO BE PROCESSED..BUT \n");
     printf("      NOTE THAT THIS FILE MUST BE AN UNFORMATTED FILE \n");
     printf("      PRODUCED AS A RESULT OF EXECUTING simulator.out \n");
     printf("      THE LAST LETTER OF WHICH MUST END IN LETTER 'X' \n");
  error:space(2);
     printf("***->");
```

```
	i=0;
	while((c=getchar()) != '\n') {
		cbuf[i]=c;
		i++;
	}
	cbuf[i]='\0';

	bufctr=i;
	for(j=0;j<bufctr;j++) {
	    gbuf[j]=fbuf[j]=cbuf[j];
	    if(cbuf[j]=='X') {
		fbuf[j]='Y';
		gbuf[j]='Z';
		gbuf[j+1]=fbuf[j+1]='\0';
		j=bufctr;
	    }
	}

	fd = open(cbuf,0);
	if (fd <= 0) {
	    printf("***->error occurred in opening file...try again");
	    space(3);
	    goto error;
	    }

	if((count = read(fd,nbytes,header)) != header)
	    printf("error occurred in nbytes read");
	ievents = (nbytes[1] - 1);
	a1 = nbytes[1];
	a = (nbytes[1]-1)*14;
	if((count = read(fd,file1,a)) != a)
	    printf("error occured in file1 read");

	if((count=read(fd,nbytes,header))!=header)
	    printf("error occurred in nbytes read");
	a2=nbytes[1];
	a=(nbytes[1]-1)*16;
	if((count=read(fd,file2,a))!=a)
	    printf("error occurred in file2 read");

	if((count=read(fd,nbytes,header))!=header)
	    printf("error occurred in header read");
	a3=nbytes[1];
	a=(nbytes[1]+1)*2;
	if((count=read(fd,iotbl,a))!=a)
	    printf("error occured in iotbl read");

	if((count=read(fd,nbytes,header))!=header)
	    printf("error occurred in header read");
	a4=nbytes[1];
	a=nbytes[1]+1;
	if((count=read(fd,names,a))!=a)
	    printf("error occurred in names read");

	close(fd);
return;
}
```

```
determine() {
     int i;
     char d,dbuf[20];

     space(2);
     printf("***->FUNCTION 'DETERMINE' ALLOWS THE USER \n");
     printf("      TO EXAMINE ALL PRIMARY DATA STRUCTURES");
    over:space(2);
     printf("***->IF THIS FEATURE IS DESIRED TYPE 1 IF NOT 0,...<RET>");
     space(2);
     printf("***->");

     i=0;
     while((d=getchar())!='\n') {
          dbuf[i]=d;
          i++;
     }
     dbuf[i]='\0';

     i=0;
     while(dbuf[i]!='\0') {
          d=dbuf[i];
          switch(d) {
               case'0':
                     scrn='0';
                     break;
               case'1':
                     scrn='1';
                     printf("***->USE CONTROL Q WHEN SCREEN FULL");
                     break;
               default:
                     printf("***->either blank or invalid entry");
                     goto over;
                     break;
          }
          i++;
     }
return;
}


display() {
  int i;

  if(scrn=='1') {
      space(2);
      bp1 = file1;
      printf("***-> FILE1 DATA STRUCTURE");
      space(2);
      printf("Infeed   #    marker    xcord   ycord   plot   length \n");
      for (i=0;i<a1; i++){
          printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t \n",
              bp1->ctrl1,bp1->nameptr, bp1->marker,bp1->xcord,
              bp1->ycord,bp1->plot,bp1->length);
          bp1++;
      }

      space(2);
      bp2 = file2;
      printf("***-> FILE2 DATA STRUCTURE");
```

```
        space(2);
        printf
        ("Infeed   trnptr  intrn   outtrn  xxcord yycord trnplot trnlen \n");
        for (i=0;i<a2; i++){
            printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t %d \t \n",
                 bp2->ctrl2,bp2->trnptr,bp2->intrn,bp2->outtrn,
                 bp2->xxcord,bp2->yycord,bp2->trnplot,bp2->trnlen);
            bp2++;
        }

        space(2);
        printf("***-> IOTBL DATA ARRAY");
        space(2);
        for (i=0; i<a3; i++) {
            printf("%d ", iotbl[i]);
        }

        space(2);
        printf("***-> NAMES DATA ARRAY");
        space(2);
        for (i=0; i<a4; i++) {
           printf("%c",names[i]);
        }
        space(2);
   }
return;
}


prepare(type) {
    /* function designates set, screen size and color table */
    int n,t,y;

    y=0;
    genisco (set);
    erase();
    screen(0.0,0.0,511.0,511.0);
    setmod(type);
    coltab();
    colort(11);
    for(n=12;n<14;n++) {
        color(n);
        for(t=0;t<512;t++) {
            segmnt(0,y+t,511,y+t);
        }
    }
return;
}


drawnode() {
    /* function displays type & location of network nodes */
    char c,*nptr,hold;
    int a,b,clrlbl,count,d,entry,h,i,j,k,l,test,x,y,*z;
    float s;
    double sqrt();

    bp1 = file1;
    a=0;
    count = 1;
```

```

    while((test=(bp1->nameptr))!=0) {
        color(14);
        test++;
        z = &names[test];
        c = *z;

        if ((b=(bp1->plot))!= 0) {
            switch (c){
                case 'I':
                    x = (bp1->xcord);
                    y = (bp1->ycord);
                    for(d=0;d<10;d++) {
                        segmnt(x-16,y+2+d,x,y+2+d);
                    }
                    linktbl[a][3]=1;
                    break;
                case 'O':
                    x = (bp1->xcord);
                    y = (bp1->ycord);
                    for(d=0;d<10;d++) {
                        segmnt(x-16,y-2-d,x,y-2-d);
                    }
                    linktbl[a][3]=1;
                    break;
                case 'R':
                    x = (bp1->xcord);
                    y = (bp1->ycord);
                    for(d=0;d<31;d++) {
                     segmnt
                     (x-18+d/4,y+15-d,x+18-d/4,y+15-d);
                    }
                    clrlbl=14;
                    label(x,y,test,clrlbl);
                    break;
                case 'S':
                    x = (bp1->xcord);
                    y = (bp1->ycord);
                    for(d=0;d<31;d++) {
                      segmnt(x-18,y-15+d,x+18,y-15+d);
                    }
                    if(vers==3)  {
                        nptr= &names[test+1];
                        hold= *nptr;
                        clrlbl=atoi(&hold);
                        label(x,y,test,clrlbl);
                        color(14);
                    }
                    else {
                         clrlbl=14;
                         label(x,y,test,clrlbl);
                    }
                    break;
                case 'T':
                    x = (bp1->xcord);
                    y = (bp1->ycord);
                    for (k=0;k<19;k++) {
                        s=k;
                        i=(x-(sqrt(324.-s*s)));
                        j=y+s;
```

```
                          l=(x+(sqrt(324.-s*s)));
                          segmnt(i,j,l,j);
                     }
                     for (k=0;k<19;k++) {
                          s=k;
                          i=(x-(sqrt(324.-s*s)));
                          j=y-s;
                          l=(x+(sqrt(324.-s*s)));
                          segmnt(i,j,l,j);
                     }
                     if(vers==3)  {
                          nptr = &names[test+1];
                          hold = *nptr;
                          clrlbl=atoi(&hold);
                          label(x,y,test,clrlbl);
                          color(14);
                     }
                     else {
                           clrlbl=14;
                           label(x,y,test,clrlbl);
                     }
                     break;
                default:
                     printf("name not valid identifier");
                     space(2);
                     break;
           }

        linktbl[a][0]=count;
        linktbl[a][1] = x;
        linktbl[a][2] = y;
        a++;
        }
    count++;
    bpl++;
    }
    if(vers==3) {
        entry=0;
        uniquetbl[entry][0]=linktbl[0][0];
        uniquetbl[entry][1]=linktbl[0][1];
        uniquetbl[entry][2]=linktbl[0][2];
        uniquetbl[entry][3]=linktbl[0][3];
        tblctr=1;
        for(i=0;i<a;i++) {
               if(uniquetbl[entry][1]==linktbl[i+1][1]  &&
                    uniquetbl[entry][2]==linktbl[i+1][2]) {
                    /* do nothing */
               }
               else {
                    entry++;
                    uniquetbl[entry][0]=linktbl[i+1][0];
                    uniquetbl[entry][1]=linktbl[i+1][1];
                    uniquetbl[entry][2]=linktbl[i+1][2];
                    uniquetbl[entry][3]=linktbl[i+1][3];
                    tblctr++;
               }
        }
    }
    nonplot=a;
return;
```

```
}


places() {
        int h,i,j,k;

        if(strn=='1') {
                space(2);
                printf("***->DATA STRUCTURE LINKTBL ");
                space(2);
                for(i=0;i<nbrplot;i++) {
                        for(h=0;h<4;h++) {
                                printf("%d --",linktbl[i][h]);
                        }
                        space(1);
                }
                space(2);
                if(vers==3) {
                        printf("***->DATA STRUCTURE UNIQUETBL");
                        space(2);
                        for(j=0;j<tblctr;j++) {
                                for(k=0;k<4;k++) {
                                        printf("%d --",uniquetbl[j][k]);
                                }
                                space(1);
                        }
                }
                space(2);
        }
return;
}


label(xx,yy,zz,clbl) {
            /* determines node label placement in relation to node */
            int f,i;
            char a;

            color(clbl);
            if(xx>250) {
                if(yy>250) a='1';
                else a='2';
                }
            else {
                if(yy>250) a='3';
                else a='4';
                }
            f=(pol->length);

            switch (a) {
                case'1':
                    for(i=0;i<2;i++) {
                        charac((xx+(20+(8*i))),yy+15,names[zz]);
                        zz++;
                    }
                    break;
                case'2':
                    for(i=0;i<2;i++) {
                        charac((xx+(20+8*i)),yy-28,names[zz]);
                        zz++;
```

```
                    }
                    break;
               case'3':
                    for(i=0;i<2;i++) {
                         charac((xx-(32-(8*i))),yy+15,names[zz]);
                         zz++;
                    }
                    break;
               case'4':
                    for (i=0;i<2;i++) {
                          charac((xx-(32-8*i)),yy-26,names[zz]);
                          zz++;
                    }
                    break;
          }
return;
}


select()  {
     int i,n;
     char v,vbuf[20];

     space(2);
     printf("***->THERE ARE 3 VERSIONS TO THIS GRAPHICS PACKAGE \n");
     printf("      PLEASE SELECT ONE OF THE FOLLOWING VERSIONS: \n");
   again:space(2);
     printf("       VERSION 1 ... PETRI-NET PACKAGE ....... TYPE 1 \n");
     printf("       VERSION 2 ... PACKET REPRESENTATION ... TYPE 2 \n");
     printf("       VERSION 3 ... MULTIROUTING PACKAGE .... TYPE 3 \n");
     n = 0;
   twice:space(2);
     printf("***->");
     n++;
     i = 0;
     while ((v=getchar()) != '\n') {
          vbuf[i] = v;
          i++;
     }
     vbuf[i] = '\0';

     i = 0;
     while(vbuf[i] != '\0')  {
          v = vbuf[i];
          if(n==1) {
                 switch(v) {
                   case '1':
                    vers = 1;
                    break;
                   case '2':
                    vers = 2;
                    break;
                   case '3':
                    vers = 3;
                    break;
                   default:
                    printf
                    ("***->incorrect version try again!");
                    goto again;
                    break;
```

```
                        }
                        i++;
                }
                else {
                        switch(v) {
                          case '0':
                           set = 0;
                           break;
                          case '1':
                           set = 1;
                           break;
                          case '2':
                           set = 2;
                           break;
                          default:
                           printf
                           ("***->incorrect genisco set try again!");
                           printf
                           ("     set selection should be 0,1 or 2");
                           n = 1;
                           goto twice;
                           break;
                        }
                        i++;
                }
        }
        if(n==1) {
                space(2);
                printf("***->NOW SELECT THE GENISCO SET YOU WISH \n");
                printf("      THE PROGRAM TO BE DISPLAYED TO..... \n");
                printf("      IN C3 LAB EITHER SET0, SET1 OR SET2 \n");
                goto twice;
        }
return;
}


imark() {
        /* marks initial state of system by calling appropriate function */
        int b,colour,e,q,x,y;

        bp1 = file1;
        dfltcolor=3;
        color(dfltcolor);
        colour=2;
        ctroverflow=0;
        while ((q=(bp1->nameptr)) !=0) {
             if((b=(bp1->plot)) !=0) {
                     switch(vers)  {
                          case 1:
                             ivers1();
                             break;
                          case 2:
                             ivers2(colour);
                             break;
                          case 3:
                             ivers3(colour);
                             break;
                     }
             }
```

```
            if(dfltcolor!=3) color(3);
            bol++;
        }
        color(14);
        printg(0,350.,480.,"TIME FRAME = 1");
        preread(1);
        trnlite(1);
        displa();
        hold();
        trnlite(2);
        reset(1);
        color(13);
        ovrflow();
        printg(0,350.,480.,"TIME FRAME = 1");
return;
}


hold() {
        int holding;

        space(2);
        printf("***->THIS IS THE INITIAL STATE OF THE NETWORK \n");
        printf("      TYPE <RETURN> TO CONTINUE EXECUTION..... \n");
        while((holding=getchar())!='\n') {
            /* do nothing loop */
        }
return;
}


ivers1() {
        int e,x,y,z;
        char check;

        e=(bol->marker);
        x=(bol->xcord);
        y=(bol->ycord);
        z=(bol->nameptr);
        if((check=names[z+1])!='I' && (check=names[z+1])!='O') {
            printg(0,x-3.0,511.-(y-3),"%d",e);
        }
        else {
            if(check=names[z+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",e);
            else printg(0,x-14.0,511.-(y-9),"%d",e);
        }
return;
}


ivers2(colour)  {
        int e,x,y,z;
        char check;

        e=(bol->marker);
        x=(bol->xcord);
        y=(bol->ycord);
        z=(bol->nameptr);
        if((check=names[z+1])!='I' && (check=names[z+1])!='O') {
            pckt2(x,y,e,colour);
```

```
	}
	else {
		if(check=names[z+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",e);
		else printg(0,x-14.0,511.-(y-9),"%d",e);
	}
return;
}


ivers3(colour) {
	char  keep,*kptr;
	int a,aa,b,bb,c,cc,clr[25],i,k,n,stack,total,x,y;

	total=0;
	n=1;
	x=(bpl->xcord);
	y=(bpl->ycord);
	a=(bpl->length);
	b=(bpl->nameptr);
	c=(bpl->marker);
	total=total+c;

	if(c>0)  {
		for(i=0;i<c;i++)  {
			kptr = &names[b+a-2];
			keep = *kptr;
			clr[n]=atoi(&keep);
			n++;
		}
	}

	kptr = &names[b+a];
	keep = *kptr;
	stack=atoi(&keep);

	if(names[b+(a-1)]!='0') {
		if(names[b+(a-1)]=='2')  stack=stack+20;
		else  stack = stack + 10;
	}

	for(i=0;i<stack-1;i++)  {
		bpl++;
		aa=(bpl->length);
		bb=(bpl->nameptr);
		cc=(bpl->marker);
		total=total+cc;
		if(cc>0)  {
			for(k=0;k<cc;k++)  {
				kptr = &names[bb+aa-2];
				keep = *kptr;
				clr[n]=atoi(&keep);
				n++;
			}
		}
	}
	if(names[b+1]!='I' && names[b+1]!='0') {
	  pckt3
	  (x,y,total,clr[1],clr[2],clr[3],clr[4],clr[5],clr[6],clr[7],
	   colour);
	}
```

```
     else {
         if(names[b+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",total);
         else printg(0,x-14.0,511.-(y-9),"%d",total);
     }
return;
}


preread(flag) {
      int bucket[2],count,fd,fg,i,nbrtrns;
      if(flag!=3) {
     if(flag==1)  {
         fd = open (fbuf, 0);
         if (fd<=0) {
             printf("***->error occurred in opening fd file");
         }
         fdfbuf=fd;
         fg = open (gbuf, 0);
         if (fg<=0) {
             printf("***->error occurred in opening fg file");
         }
         fdgbuf=fg;
     }
     if((count=read (fdfbuf, bucket, 2))!=2) {
         printf("***->error occurred in fd bucket read");
     }
     if((count=read (fdfbuf, buffer,(ievents*2)))!=(ievents*2)) {
         printf("***->error occurred in buffer read");
     }
     if((count=read (fdgbuf, bucket, 2))!=2) {
         printf("***->error occurred in fg bucket read");
     }
     if((count=read (fdgbuf, cntrl,2))!=2) {
         printf("***->error occurred in cntrl read");
     }
     if(cntrl[0]==0) {
         space(2);
         printf("***->the last network state has been achieved");
         kpictures=pictures+1;
         space(2);
     }
     else {
         if((count=read (fdgbuf, bucket, 2))!=2) {
               printf("***->error occurred in bucket read");
         }
         nbrtrns = cntrl[0]*2;
         if((count=read (fdgbuf, firing,nbrtrns))!= nbrtrns) {
               printf("***->error occurred in firing read");
         }
     }
      }
      else {
     close(fdfbuf);
     close(fdgbuf);
      }
return;
}


stage() {
```

```
	if (kpictures!=0 && kpictures<pictures+1) {
		preread(2);
		if(kpictures!=pictures+1) {
			trnlite(1);
			displa();
			pause(1);
			trnlite(2);
		}
	}
return;
}


marking() {
	/* function displays successive iterations of the network */
	int colour,draw,i,mark,n,x,y;

	bpl = file1;
	n = 2;

	/* following loop processes ievent # data entries each pass */
	for(kpictures=0;kpictures<pictures;kpictures++) {
		stage();
		if(kpictures>0) {
			reset();
			color(13);
			ovrflow();
			printg(0,350.,480.,"TIME FRAME = %d",n);
			n++;
		}
		iflag = kpictures;
		draw = (bpl->plot);
		dfltcolor=3;
		color (dfltcolor);
		colour=2;
		ctroverflow=0;
		for (i=0; i < ievents; i++)  {
			if (draw == 1) {
				ictr = i;
				switch (vers)  {
					case 1:
						vers1();
						break;
					case 2:
						vers2(colour);
						break;
					case 3:
						vers3(colour);
						break;
				}
			}
			i = ictr++;
			if(dfltcolor!=3) color(3);
			bpl++;
			draw = (bpl -> plot);
		}
		dfltcolor=14;
		color(dfltcolor);
		colour=13;
```

```
            printg(0,350.,480.,"TIME FRAME = %d",n);
            pause(2);
            trnlite(3);
            bpl = filel;
        }
        preread(3);
return;
}


ovrflow() {

        int i,x,y;

        i=0;
        while(overflowtbl[i][0]!=0) {
            x=overflowtbl[i][0];
            y=overflowtbl[i][1];
            block((x-3)*1.,511.-(y+28)*1.,(x+10)*1.,511.-(y+20)*1.);
            overflowtbl[i][0]=0;
            overflowtbl[i][1]=0;
            i++;
        }
return;
}


vers1() {
        int e,x,y,z;
        char check;

        color(3);
        e=buffer[ictr];
        x=(bpl->xcord);
        y=(bpl->ycord);
        z=(bpl->nameptr);
        if((check=names[z+1])!='I' && (check=names[z+1])!='O') {
             printg(0,x-3.0,511.-(y-3),"%d",e);
        }
        else {
             if(check=names[z+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",e);
             else printg(0,x-14.0,511.-(y-9),"%d",e);
        }
return;
}


vers2(colour)  {
        int mark,x,y,z;
        char check;

        x=(bpl->xcord);
        y=(bpl->ycord);
        z=(bpl->nameptr);
        mark=buffer[ictr];
        if((check=names[z+1])!='I' && (check=names[z+1])!='O') {
         pckt2(x,y,mark,colour);
        }
        else {
         if((check=names[z+1])=='I') printg(0,x-14.0,511.-(y+2),"%d",mark);
```

```
      else printg(0,x-14.0,511.-(y-9),"%d",mark);
     }
return;
}


vers3(colour)  {
     int a,aa,b,bb,mark,marks,clr[25],j,k,n,stack,total,x,y;
     char keep,*kptr;

     total=0;
     n=1;
     x=(bpl->xcord);
     y=(bpl->ycord);
     a=(bpl->length);
     b=(bpl->nameptr);
     mark=buffer[ictr];
     total=total+mark;
     if(mark>0)  {
          for(j=0;j<mark;j++)  {
               kptr = &names[b+a-2];
               keep = *kptr;
               clr[n]=atoi(&keep);
               n++;
          }
     }

     kptr = &names[b+a];
     keep = *kptr;
     stack=atoi(&keep);

     if(names[b+(a-1)]!='0') {
          if(names[b+(a-1)]=='2')  stack=stack+20;
          else  stack = stack + 10;
     }

     for(j=0;j<stack-1;j++)  {
          bpl++;
          ictr++;
           aa=(bpl->length);
           bb=(bpl->nameptr);
           marks=buffer[ictr];
           total=total+marks;
           if(marks>0)  {
                for(k=0;k<marks;k++)  {
                     kptr = &names[bb+aa-2];
                     keep = *kptr;
                     clr[n]=atoi(&keep);
                     n++;
                }
           }
      }
      if(names[b+1]!='I' && names[b+1]!='0') {
        pckt3
        (x,y,total,clr[1],clr[2],clr[3],clr[4],clr[5],clr[6],clr[7],
         colour);
      }
      else {
          if(names[b+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",total);
          else printg(0,x-14.0,511.-(y-9),"%d",total);
```

```
     }
return;
}


pckt2(xaxis,yaxis,point,class) {

 switch(point) {
   case 0:
    break;
   case 1:
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    break;
   case 2:
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    break;
   case 3:
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
    break;
   case 4:
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
    block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
    break;
   case 5:
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
    block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
    block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
    break;
   case 6:
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
    block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
    block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
    block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
    break;
   case 7:
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
    block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
    block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
    block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
    block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
    break;
   default:
    overflowtbl[ctroverflow][0]=xaxis;
    overflowtbl[ctroverflow][1]=yaxis;
    ctroverflow++;
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
    block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
```

```
      block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
      block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
      block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
      color(class);
      dfltcolor=class;
      printq(0,xaxis-3.0,511.-(yaxis+22)+3.0,"%d",point-7);
      break;
 }
return;
}



pckt3(xaxis,yaxis,total,c1,c2,c3,c4,c5,c6,c7,class)  {

 switch (total)  {
  case 0:
    break;
  case 1:
    color(c1);
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    break;
  case 2:
    color(c1);
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    color(c2);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    break;
  case 3:
    color(c1);
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    color(c2);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    color(c3);
    block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
    break;
  case 4:
    color(c1);
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    color(c2);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    color(c3);
    block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
    color(c4);
    block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
    break;
  case 5:
    color(c1);
    block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
    color(c2);
    block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
    color(c3);
    block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
    color(c4);
    block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
    color(c5);
    block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
    break;
  case 6:
    color(c1);
```

```
      block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
      color(c2);
      block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
      color(c3);
      block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
      color(c4);
      block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
      color(c5);
      block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
      color(c6);
      block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
      break;
    case 7:
      color(c1);
      block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
      color(c2);
      block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
      color(c3);
      block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
      color(c4);
      block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
      color(c5);
      block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
      color(c6);
      block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
      color(c7);
      block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
      break;
    default:
      overflowtbl[ctroverflow][0]=xaxis;
      overflowtbl[ctroverflow][1]=yaxis;
      ctroverflow++;
      color(c1);
      block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
      color(c2);
      block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
      color(c3);
      block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
      color(c4);
      block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
      color(c5);
      block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
      color(c6);
      block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
      color(c7);
      block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
      color(class);
      dfltcolor=class;
      printg(0,xaxis-3.0,511.-(yaxis+22)+3.0,"%d",total-7);
      break;
    }
 return;
 }


 reset() {
      /* reset function for successive network iterations */
      int i,mark,x,y,z;

      if(vers==1 || vers==2) {
```

```
              for(i=0;i<nbrplot;i++)  {
               color(14);
               x=linktbl[i][1];
               y=linktbl[i][2];
               z=linktbl[i][3];
               if(z==0) {
               block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
               block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
               }
               else {
               block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
               block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
               }
              }
       }
       else {
              for(i=0;i<tblctr-1;i++) {
               color(14);
               x=uniquetbl[i][1];
               y=uniquetbl[i][2];
               z=uniquetbl[i][3];
               if(z==0) {
               block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
               block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
               }
               else {
                    block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
                    block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
               }
              }
       }
return;
}


trnlite(tflag) {
      int flag,h,i,inp,j,k,l,m,n,on,outp,ptr,r,s,x,xx,y,yy;

      for(h=0;h<cntrl[0];h++) {
     bp2=file2;
     for(j=0;j<(firing[h]-1);j++) {
         bp2++;
     }
     if(on=(bp2->trnplot)==1) {
         x=(bp2->xxcord);
         y=(bp2->yycord);
         if(tflag==1 || tflag==2) color(11);
         else color(15);
         segmnt(x,y-20,x,y+20);
         l=(bp2->trnlen);
         k=(bp2->trnptr);
         for(i=0;i<l;i++) {
              charac((x-8*(8+i)),y+24,names[k+1]);
              k++;
         }

          if(tflag==1 || tflag==3) {
         if(tflag==1) color(11);
         else color(14);
         flag=0;
```

```
          m=(bp2->intrn);
          r=iotbl[m];
          for(i=0;i<r;i++) {
               inp=iotbl[m+1];
               for(j=0;j<nbrplot;j++) {
                    if(inp==linktbl[j][0]) {
                         xx=linktbl[j][1];
                         yy=linktbl[j][2];
                         if(flag==0) {
                                   if(xx<=x) flag=1;
                                   else flag=2;
                                   sflag[h]=flag;
                         }
                         if(flag==1) intrns1(xx,yy,x,y);
                         else intrns2(xx,yy,x,y);
                         j=nbrplot;
                    }
               }
               m++;
          }
           }

           if(tflag==2 || tflag==3) {
          if(tflag==2) color(11);
          else color(14);
          n=(bp2->outtrn);
          s=iotbl[n];
          for(i=0;i<s;i++) {
               outp=iotbl[n+1];
               for(j=0;j<nbrplot;j++) {
                    if(outp==linktbl[j][0]) {
                         xx=linktbl[j][1];
                         yy=linktbl[j][2];
                         if(sflag[h]==1) outrns1(xx,yy,x,y);
                         else outrns2(xx,yy,x,y);
                         j=nbrplot;
                     }
               }
               n++;
          }
           }
     }
      }
return;
}


trnslink() {
      int flag,i,inp,j,k,l,m,n,on,outp,ptr,r,s,x,xx,y,yy;

      bp2=file2;
      while(ptr=(bp2->trnptr)!=0) {
     if(on=(bp2->trnplot)==1) {
          x=(bp2->xxcord);
          y=(bp2->yycord);
          color(15);
          segmnt(x,y-20,x,y+20);
          l=(bp2->trnlen);
          k=(bp2->trnptr);
          for(i=0;i<l;i++) {
```

```
                    charac((x-8+(8*i)),y+24,names[k+1]);
                    k++;
          }

          color(14);
          flag=0;
          m=(bp2->intrn);
          r=iotbl[m];
          for(i=0;i<r;i++) {
               inp=iotbl[m+1];
               for(j=0;j<nbrplot;j++) {
                    if(inp==linktbl[j][0]) {
                         xx=linktbl[j][1];
                         yy=linktbl[j][2];
                         if(flag==0) {
                                   if(xx<=x) flag=1;
                                   else flag=2;
                         }
                         if(flag==1) intrns1(xx,yy,x,y);
                         else intrns2(xx,yy,x,y);
                         j=nbrplot;
                    }
               }
               m++;
          }

          n=(bp2->outtrn);
          s=iotbl[n];
          for(i=0;i<s;i++) {
               outp=iotbl[n+1];
               for(j=0;j<nbrplot;j++) {
                    if(outp==linktbl[j][0]) {
                         xx=linktbl[j][1];
                         yy=linktbl[j][2];
                         if(flag==1) outrns1(xx,yy,x,y);
                         else outrns2(xx,yy,x,y);
                         j=nbrplot;
                    }
               }
               n++;
          }
     }
      bp2++;
      }
return;
}


intrns1(xx,yy,x,y) {
     int input;

     if(xx<(x-12)) {
          if(yy<=y) {
               if(yy==y) input=0;
               else input=1;
          }
          else input=2;
     }
     else {
          if(yy<=y) input=4;
```

```
          else inout=3;
     }

     switch (inout) {
          case 0:
               segmnt(xx+16,yy,x-1,y);
               rghtarrow(x-1,y);
               break;
          case 1:
               segmnt(xx+12,yy+12,x-12,y-8);
               segmnt(x-12,y-8,x-1,y-8);
               rghtarrow(x-1,y-8);
               break;
          case 2:
               segmnt(xx+12,yy-12,x-12,y+8);
               segmnt(x-12,y+8,x-1,y+8);
               rghtarrow(x-1,y+8);
               break;
          case 3:
               segmnt(xx-12,yy-12,x-12,y+36);
               segmnt(x-12,y+16,x-1,y+16);
               lftsemi(x-12,y+26);
               rghtarrow(x-1,y+16);
               break;
          case 4:
               segmnt(xx-12,yy+12,x-12,y-36);
               segmnt(x-12,y-16,x-1,y-16);
               lftsemi(x-12,y-26);
               rghtarrow(x-1,y-16);
               break;
     }
return;
}


rghtarrow(x,y) {

     segmnt(x-4,y-4,x,y);
     segmnt(x-4,y+4,x,y);
return;
}


lftsemi(x,y) {
     int i,j,k;
     double sqrt();

     for(k=0;k<11;k++){
          i=(y-(sqrt(100.-k*k)));
          j=(y+(sqrt(100.-k*k)));
          dot(x-(k/2.),511.-i);
          dot(x-(k/2.),511.-j);
          if(k>8)  {
               dot(x-(k/2.),511.-i-2);
               dot(x-(k/2.),511.-i-1);
               dot(x-(k/2.),511.-i+1);
               dot(x-(k/2.),511.-i+2);
               dot(x-(k/2.),511.-j-2);
               dot(x-(k/2.),511.-j-1);
               dot(x-(k/2.),511.-j+1);
```

```
				dot(x-(k/2.),511.-j+2);
			}
		}
	return;
	}


	outrnsl(xx,yy,x,y) {
		int output;

		if(xx>(x+12)) {
			if(yy<=y) {
				if(yy==y) output=0;
				else output=1;
			}
			else output=2;
		}
		else {
			if(yy<=y) output=4;
			else output=3;
		}

		switch (output) {
			case 0:
				segmnt(x+1,y,xx-20,yy);
				rghtarrow(xx-20,yy);
				break;
			case 1:
				segmnt(x+12,y-8,xx-28,yy+8);
				segmnt(xx-28,yy+8,xx-20,yy+8);
				rghtarrow(xx-20,yy+8);
				segmnt(x+1,y-8,x+12,y-8);
				break;
			case 2:
				segmnt(x+12,y+8,xx-28,yy-8);
				segmnt(xx-28,yy-8,xx-20,yy-8);
				rghtarrow(xx-20,yy-8);
				segmnt(x+1,y+8,x+12,y+8);
				break;
			case 3:
				segmnt(x+1,y+16,x+12,y+16);
				segmnt(x+12,y+36,xx+28,yy-8);
				segmnt(xx+28,yy-8,xx+20,yy-8);
				lftarrow(xx+20,yy-8);
				rghtsemi(x+12,y+26);
				break;
			case 4:
				segmnt(x+1,y-16,x+12,y-16);
				segmnt(x+12,y-36,xx+28,yy+8);
				segmnt(xx+28,yy+8,xx+20,yy+8);
				lftarrow(xx+20,yy+8);
				rghtsemi(x+12,y-26);
				break;
			}
	return;
	}


	rghtsemi(x,y)  {
		int i,j,k;
```

```
	double sqrt();

	for (k=0;k<11;k++) {
		i = (y-(sqrt(100.-k*k)));
		j = (y+(sqrt(100.-k*k)));
		dot(x+(k/2.),511.-i);
		dot(x+(k/2.),511.-j);
		if(k>8) {
			dot(x+(k/2.),511.-i-2);
			dot(x+(k/2.),511.-i-1);
			dot(x+(k/2.),511.-i+1);
			dot(x+(k/2.),511.-i+2);
			dot(x+(k/2.),511.-j-2);
			dot(x+(k/2.),511.-j-1);
			dot(x+(k/2.),511.-j+1);
			dot(x+(k/2.),511.-j+2);
		}
	}
return;
}


lftarrow(x,y)  {

	segmnt(x+4,y-4,x,y);
	segmnt(x+4,y+4,x,y);
return;
}


intrns2(xx,yy,x,y) {
	int input;

	if(xx>(x+12)) {
		if(yy<=y) {
			if(yy==y) input=0;
			else input=1;
		}
		else input=2;
	}
	else {
		if(yy<=y) input=4;
		else input=3;
	}

	switch (input) {
		case 0:
			segmnt(xx-16,yy,x+1,y);
			lftarrow(x+1,y);
			break;
		case 1:
			segmnt(xx-12,yy+12,x+12,y-8);
			segmnt(x+12,y-8,x+1,y-8);
			lftarrow(x+1,y-8);
			break;
		case 2:
			segmnt(xx-12,yy-12,x+12,y+8);
			segmnt(x+12,y+8,x+1,y+8);
			lftarrow(x+1,y+8);
			break;
```

```
		case 3:
			segmnt(xx+12,yy-12,x+12,y+36);
			segmnt(x+12,y+16,x+1,y+16);
			rghtsemi(x+12,y+26);
			lftarrow(x+1,y+16);
			break;
		case 4:
			segmnt(xx+12,yy+12,x+12,y-36);
			segmnt(x+12,y-16,x+1,y-16);
			rghtsemi(x+12,y-26);
			lftarrow(x+1,y-16);
			break;
	}
return;
}


outrns2(xx,yy,x,y) {
	int output;

	if(xx<(x-12)) {
		if(yy<=y) {
			if(yy==y) output=0;
			else output=1;
		}
		else output=2;
	}
	else {
		if(yy<=y) output=4;
		else output=3;
	}

	switch (output) {
		case 0:
			segmnt(x-1,y,xx+20,yy);
			lftarrow(xx+20,yy);
			break;
		case 1:
			segmnt(x-12,y-8,xx+28,yy+8);
			segmnt(xx+28,yy+8,xx+20,yy+8);
			lftarrow(xx+20,yy+8);
			segmnt(x-1,y-8,x-12,y-8);
			break;
		case 2:
			segmnt(x-12,y+8,xx+28,yy-8);
			segmnt(xx+28,yy-8,xx+20,yy-8);
			lftarrow(xx+20,yy-8);
			segmnt(x-1,y+8,x-12,y+8);
			break;
		case 3:
			segmnt(x-1,y+16,x-12,y+16);
			segmnt(x-12,y+36,xx-28,yy-8);
			segmnt(xx-28,yy-8,xx-20,yy-8);
			rghtarrow(xx-20,yy-8);
			lftsemi(x-12,y+26);
			break;
		case 4:
			segmnt(x-1,y-16,x-12,y-16);
			segmnt(x-12,y-36,xx-28,yy+8);
			segmnt(xx-28,yy+8,xx-20,yy+8);
```

```
                rghtarrow(xx-20,yy+8);
                lftsemi(x-12,y-26);
                break;
        }
return;
}


coltab()  {
    int i;

    i=11*16;
    lodcol(i++,15,15,15);   /*  color 0   */
    lodcol(i++,0,10,0);     /*  color 1   */
    lodcol(i++,15,0,0);     /*  color 2   */
    lodcol(i++,15,15,0);    /*  color 3   */
    lodcol(i++,12,0,12);    /*  color 4   */
    lodcol(i++,5,5,12);     /*  color 5   */
    lodcol(i++,6,6,5);      /*  color 6   */
    lodcol(i++,5,3,3);      /*  color 7   */
    lodcol(i++,10,o,10);    /*  color 8   */
    lodcol(i++,12,5,5);     /*  color 9   */
    lodcol(i++,5,5,3);      /*  color 10 */
    lodcol(i++,12,12,0);    /*  color 11 */
    lodcol(i++,8,7,3);      /*  color 12 */
    lodcol(i++,5,4,2);      /*  color 13 */
    lodcol(i++,0,0,6);      /*  color 14 */
    lodcol(i++,6,0,0);      /*  color 15 */
return;
}


        /**************************************************************/
        /*********                                          **********/
        /*********   END OF PROGRAM TRANSGRAPH.C           **********/
        /*********                                          **********/
        /**************************************************************/
```

```
#

        /**************************************************************/
        /**************************************************************/
        /*********                                           *********/
        /*********           PROGRAM LINKGRAPH.C             *********/
        /*********                                           *********/
        /*********      STEPHEN C. JENNINGS JC91 USMC        *********/
        /*********        ROBERT J. HARTEL CS91 USA          *********/
        /*********                                           *********/
        /*********        WRITTEN FALL QUARTER 1979          *********/
        /*********        NAVAL POSTGRADUATE SCHOOL          *********/
        /*********          MONTEREY, CALIFORNIA             *********/
        /*********                                           *********/
        /**************************************************************/
        /**************************************************************/


/*****************************************/
/****      EXTERNAL DECLARATIONS     ****/
/*****************************************/


/**** LITERALS ****/

#define  header     4      /* contains control # as to bytes read in ... */
#define  std        10     /* the standard input read buffer length .... */
#define  fired      100    /* the max # of transitions fired in 1 frame  */
#define  frames     150    /* indicates the total # of network states .. */
#define  bounds     360    /* bounds on max # of nodes or transitions .. */
#define  upper      2000   /* defines iotbl max length ................. */
#define  limit      3600   /* defines names max length ................. */


/**** STRUCTURES ****/

struct {                   /* data structure information on net nodes .. */

     int ctrl1;          /* store control char not used in program ... */
     int nameptr;        /* index to names array ..................... */
     int marker;         /* initial marker state of the network ...... */
     int xcord;          /* x cordinate of place ..................... */
     int ycord;          /* y cordinate of place ..................... */
     int plot;           /* whether or not place is to be plotted .... */
     int length;         /* length of name associated with place ..... */

}file1 [bounds],  *hp1;    /* pointer into data structure .............. */


struct {                   /* data structure information on transitions. */

     int ctrl2;          /* store control char not used in program ... */
     int trnptr;         /* index to names array ..................... */
     int intrn;          /* pointer to inputs for a transition ....... */
     int outtrn;         /* pointer to outputs for a transition ...... */
     int xxcord;         /* x cordinate of transition ................ */
     int yycord;         /* y cordinate of transition ................ */
     int trnplot;        /* whether or not transition is to be plotted */
     int trnlen;         /* length of name associated with transition  */

```

```
}file2 [bounds],  *bp2;     /* pointer into data structure .............. */


/**** INTEGERS ****/

int a1,a2,a3,a4;            /* global storage for each data structure ... */
int buffer[bounds];         /* buffer into which each frame is read ..... */
int cntrl[1];               /* variable containing # transitions fired .. */
int ctroverflow;            /* keep track of nodes overflow status ...... */
int dfltcolor;              /* a default color for indicating overflows . */
int fdfbuf;                 /* file descriptor for RUN--Y files ......... */
int fdgbuf;                 /* file descriptor for RUN--Z files ......... */
int firing[bounds];         /* storage into which fired places are read . */
int ictr;                   /* counter passed to a function ............. */
int ievents;                /* number of non- & displayable nodes ....... */
int iflag;                  /* counter for the interrupt mechanism ...... */
int iotbl[upper];           /* forms input-to-output relationship ....... */
int ktnframe;               /* counter for the iterations of the network. */
int linktbl[bounds][4];     /* version 1 & 2 screen nodes locations ..... */
int nbrplot;                /* counter for number of displayed nodes .... */
int nbytes[2];              /* store count fields for data structures ... */
int overflowtbl[fired][2];  /* data structure to store overflow locations */
int set;                    /* user selected conrac graphics screen ..... */
int tabcount;               /* counter for link revert .................. */
int tblctr;                 /* a counter for version 3..reset conditions. */
int uniquetbl[fired][4];    /* reset table locations for version 3 ...... */
int vers;                   /* user selected option ..................... */
int xinstore[fired];        /* coordinates retained for link revert ..... */
int xoutstore[fired];       /* coordinates retained for link revert ..... */
int yinstore[fired];        /* coordinates retained for link revert ..... */
int youtstore[fired];       /* coordinates retained for link revert ..... */


/**** CHARACTERS ****/

char cbuf[std];             /* buffer store of file to be executed ...... */
char dbuf[std];             /* buffer store determining data scan ....... */
char fbuf[std];             /* buffer to store name of second file ...... */
char gbuf[std];             /* buffer to store name of third file........ */
char names[limit];          /* character array for node labels .......... */
char scrn;                  /* option variable for display to the screen. */
char tbuf[std];             /* buffer to store timing variable .......... */
char vbuf[std];             /* buffer to store version selected ......... */
char timing;                /* variable to set program execution timing . */


/*********************************/
/****     FUNCTION MAIN     ****/
/*********************************/


main() {

     extern rubout();   /* declare 'rubout' globally ................ */
     init();            /* read input file .......................... */
     determine();       /* verify if user wants to see data structure */
     display();         /* display input to crt ..................... */
     select();          /* select version of simulation & genisco set */
     prepare(2);        /* prepare genesco-conrac ................... */
     drawnode();        /* draw network nodes on conrac ............. */
```

```
    places();           /* verify correct nodes drawn ............... */
    link();             /* function depicts network connectivity .... */
    imark();            /* starting status of network packets ....... */
    signal(2,rubout);   /* sets 'BRK' as interrupt .................. */
    marking();          /* successive iterations of network flow .... */
    gnfini();           /* closing out graphics facilities .......... */

}


/*************************************/
/****      PROGRAM  FUNCTIONS     ****/
/*************************************/


pause(peroid) {
    /* function necessary as sleep() not compatible with signal() */
    int i,j,k;

    printf("***>>interrupt......");
    for(i=0;i<peroid;i++) {
        for(j=0;j<400;j++) {
            for(k=0;k<1000;k++) {
            }
        }
    }
    printf("***>>wait......");
return;
}


rubout()  {
    /* function enables the 'brk' key as the interrupt signal */
    char halt;

    space(2);
    printf
    ("***>>> received signal...frame number %d...<ret> to continue \n",
    (iflag+2));
    printf
    ("***>>> for termination of program...type 'brk' from console \n");
    while ((halt=getchar())!='\n')  {
          /*  do-nothing loop  */
    }
    signal(2,rubout);
return;
}


space(returns) {
     int i;

     for (i=0; i<returns; i++) {
         printf("\n");
         }
return;
}


init() {
```

```
    /* function opens unformatted file & initializes start condition */
    int a,bufctr,count,fd,i,j;
    char c,f;

    space(2);
    printf("***->LINKGRAPH ILLUSTRATES NETWORK SIMULATION MODELS");
    space(2);
    printf("***->ENTER THE NAME OF THE FILE TO BE PROCESSED..BUT \n");
    printf("      NOTE THAT THIS FILE MUST BE AN UNFORMATTED FILE \n");
    printf("      PRODUCED AS A RESULT OF EXECUTING simulator.out \n");
    printf("      THE LAST LETTER OF WHICH MUST END IN LETTER 'X' \n");
error:space(2);
    printf("***->");

    i=0;
    while((c=getchar()) != '\n') {
          cbuf[i]=c;
          i++;
    }
    cbuf[i]='\0';

    bufctr=i;
    for(j=0;j<bufctr;j++) {
         qbuf[j]=fbuf[j]=cbuf[j];
         if(cbuf[j]=='X') {
              fbuf[j]='Y';
              qbuf[j]='Z';
              qbuf[j+1]=fbuf[j+1]='\0';
              j=bufctr;
         }
    }

    fd = open(cbuf,0);
    if (fd <= 0) {
         printf
         ("***->error occurred in opening file......try again");
         space(3);
         goto error;
         }

    if((count = read(fd,nbytes,header)) != header)
         printf("error occurred in nbytes read");
    ievents = (nbytes[1] - 1);
    a1 = nbytes[1];
    a = (nbytes[1]-1)*14;
    if((count = read(fd,file1,a)) != a)
         printf("error occured in file1 read");

    if((count=read(fd,nbytes,header))!=header)
         printf("error occurred in nbytes read");
    a2=nbytes[1];
    a=(nbytes[1]-1)*16;
    if((count=read(fd,file2,a))!=a)
         printf("error occurred in file2 read");

    if((count=read(fd,nbytes,header))!=header)
         printf("error occurred in header read");
    a3=nbytes[1];
    a=(nbytes[1]+1)*2;
    if((count=read(fd,iotbl,a))!=a)
```

```
            printf("error occured in iotbl read");

        if((count=read(fd,nbytes,header))!=header)
            printf("error occurred in header read");
        a4=nbytes[1];
        a=nbytes[1]+1;
        if((count=read(fd,names,a))!=a)
            printf("error occurred in names read");

        close(fd);
return;
}


determine() {
        int i;
        char d;

        space(2);
        printf("***->FUNCTION 'DETERMINE' ALLOWS THE USER \n");
        printf("      TO EXAMINE ALL PRIMARY DATA STRUCTURES");
      over:space(2);
        printf
        ("***->IF THIS FEATURE IS DESIRED TYPE 1 IF NOT 0, THEN <RET>");
        space(2);
        printf("***->");

        i=0;
        while((d=getchar())!='\n') {
            dbuf[i]=d;
            i++;
        }
        dbuf[i]='\0';

        i=0;
        while(dbuf[i]!='\0') {
            d=dbuf[i];
            switch(d) {
                case'0':
                        scrn='0';
                        break;
                case'1':
                        scrn='1';
                        printf("***->USE CONTROL Q WHEN SCREEN FULL");
                        break;
                default:
                        printf("***->either blank or invalid entry");
                        goto over;
                        break;
            }
            i++;
        }
return;
}


display() {
      int i;

      if(scrn=='1') {
```

```
        space(2);
        bp1 = file1;
        printf("***-> FILE1 DATA STRUCTURE");
        space(2);
        printf
        ("Infeed   #    marker    xcord   ycord   plot   length \n");
        for (i=0;i<a1; i++){
            printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t \n",
               bp1->ctrl1,bp1->nameptr, bp1->marker,bp1->xcord,
               bp1->ycord,bp1->plot,bp1->length);
            bp1++;
        }

        space(2);
        bp2 = file2;
        printf("***-> FILE2 DATA STRUCTURE");
        space(2);
        printf
        ("Infeed   trnptr  intrn   outtrn  xxcord yycord trnplot trnlen \n");
        for (i=0;i<a2; i++){
            printf("%d \t %d \t %d \t %d \t %d \t %d \t %d \t %d \t \n",
                 bp2->ctrl2,bp2->trnptr,bp2->intrn,bp2->outtrn,
                 bp2->xxcord,bp2->yycord,bp2->trnplot,bp2->trnlen);
            bp2++;
        }

        space(2);
        printf("***-> IOTBL DATA ARRAY");
        space(2);
        for (i=0; i<a3; i++) {
            printf("%d ", iotbl[i]);
        }

        space(2);
        printf("***-> NAMES DATA ARRAY");
        space(2);
        for (i=0; i<a4+1; i++) {
           printf("%c",names[i]);
        }
        space(2);
    }
return;
}


prepare(type) {
     /* function designates set, screen size and color table */
     int n,t,y;

     y=0;
     genisco (set);
     erase();
     screen(0.0,0.0,511.0,511.0);
     setmod(type);
     coltab();
     colort(11);
     for(n=12;n<14;n++) {
         color(n);
         for(t=0;t<512;t++) {
                segmnt(0,y+t,511,y+t);
```

```
          }
     }
return;
}


drawnode() {
     /* function displays type & location of network nodes */
     char c,*nptr,hold;
     int a,b,clrlbl,count,d,entry,h,i,j,k,l,node,test,x,y,*z;
     float s;
     double sqrt();

     bpl = file1;
     a=0;
     count = 1;

     while((test=(bpl->nameptr))!=0) {
          color(14);
          test++;
          z = &names[test];
          c = *z;

          if ((b=(bpl->plot))!= 0) {
               switch (c){
                    case 'I':
                         node=1;
                         x = (bpl->xcord);
                         y = (bpl->ycord);
                         for(d=0;d<10;d++) {
                              segmnt(x-16,y+2+d,x,y+2+d);
                         }
                         if(vers==3)  {
                              nptr= &names[test+1];
                              hold= *nptr;
                              clrlbl=atoi(&hold);
                              label(x,y,test,clrlbl,node);
                              color(14);
                         }
                         else {
                              clrlbl=14;
                              label(x,y,test,clrlbl,node);
                         }
                         linktbl[a][3]=1;
                         break;
                    case 'O':
                         node=2;
                         x = (bpl->xcord);
                         y = (bpl->ycord);
                         for(d=0;d<10;d++) {
                              segmnt(x-16,y-2-d,x,y-2-d);
                         }
                         if(vers==3)  {
                              nptr= &names[test+1];
                              hold= *nptr;
                              clrlbl=atoi(&hold);
                              label(x,y,test,clrlbl,node);
                              color(14);
                         }
                         else {
```

```
                        clrlbl=14;
                        label(x,y,test,clrlbl,node);
                    }
                    linktbl[a][3]=1;
                    break;
            case 'R':
                    node=0;
                    x = (bpl->xcord);
                    y = (bpl->ycord);
                    for(d=0;d<31;d++) {
                    segmnt
                    (x-18+d/4,y+15-d,x+18-d/4,y+15-d);
                    }
                    clrlbl=14;
                    label(x,y,test,clrlbl,node);
                    break;
            case 'S':
                    node=0;
                    x = (bpl->xcord);
                    y = (bpl->ycord);
                    for(d=0;d<31;d++) {
                      segmnt(x-18,y-15+d,x+18,y-15+d);
                    }
                    if(vers==3)  {
                           nptr= &names[test+1];
                           hold= *nptr;
                           clrlbl=atoi(&hold);
                           label(x,y,test,clrlbl,node);
                           color(14);
                    }
                    else {
                           clrlbl=14;
                           label(x,v,test,clrlbl,node);
                    }
                    break;
            case 'T':
                    node=0;
                    x = (bpl->xcord);
                    y = (bpl->ycord);
                    for (k=0;k<19;k++) {
                         s=k;
                         i=(x-(sqrt(324.-s*s)));
                         j=y+s;
                         l=(x+(sqrt(324.-s*s)));
                         segmnt(i,j,l,j);
                    }
                    for (k=0;k<19;k++) {
                         s=k;
                         i=(x-(sqrt(324.-s*s)));
                         j=y-s;
                         l=(x+(sqrt(324.-s*s)));
                         segmnt(i,j,l,j);
                    }
                    if(vers==3)  {
                         nptr = &names[test+1];
                         hold = *nptr;
                         clrlbl=atoi(&hold);
                         label(x,y,test,clrlbl,node);
                         color(14);
                    }
```

```
                        else {
                              clrlbl=14;
                              label(x,y,test,clrlbl,node);
                        }
                        break;
                  default:
                        printf
                        ("name not valid identifier");
                        space(2);
                        break;
            }

          linktbl[a][0]=count;
          linktbl[a][1] = x;
          linktbl[a][2] = y;
          a++;
          }
     count++;
     bpl++;
     }
     if(vers==3) {
          entry=0;
          uniquetbl[entry][0]=linktbl[0][0];
          uniquetbl[entry][1]=linktbl[0][1];
          uniquetbl[entry][2]=linktbl[0][2];
          uniquetbl[entry][3]=linktbl[0][3];
          tblctr=1;
          for(i=0;i<a;i++) {
                  if(uniquetbl[entry][1]==linktbl[i+1][1]  &&
                       uniquetbl[entry][2]==linktbl[i+1][2]) {
                       /* do nothing */
                  }
                  else {
                       entry++;
                       uniquetbl[entry][0]=linktbl[i+1][0];
                       uniquetbl[entry][1]=linktbl[i+1][1];
                       uniquetbl[entry][2]=linktbl[i+1][2];
                       uniquetbl[entry][3]=linktbl[i+1][3];
                       tblctr++;
                  }
          }
     }
     nbrplot=a;
return;
}


places() {
     int h,i,j,k;

     if(scrn=='1') {
          space(2);
          printf("***->DATA STRUCTURE LINKTBL ");
          space(2);
          for(i=0;i<nbrplot;i++) {
               for(h=0;h<4;h++) {
                    printf("%d --",linktbl[i][h]);
               }
               space(1);
          }
```

```
        space(2);
        if(vers==3) {
            printf("***->DATA STRUCTURE UNIQUETBL");
            space(2);
            for(j=0;j<tblctr;j++) {
                for(k=0;k<4;k++) {
                    printf("%d --",uniquetbl[j][k]);
                }
                space(1);
            }
        }
        space(2);
    }
return;
}


label(xx,yy,zz,clbl,place) {
       /* determines node label placement in relation to node */
       int f,i;

       color(clbl);

       switch (place) {
           case 0 :
            if(xx>250) {
               for(i=0;i<2;i++) {
                  charac((xx+(22+(8*i))),yy,names[zz]);
                  zz++;
               }
            }
            else {
               for(i=0;i<2;i++) {
                  charac((xx-(34-(8*i))),yy,names[zz]);
                  zz++;
               }
            }
            break;
           case 1 :
            if(xx>250) {
               for(i=0;i<2;i++) {
                  charac((xx+4+(8*i)),yy+2,names[zz]);
                  zz++;
               }
            }
            else {
               for(i=0;i<2;i++) {
                  charac((xx-(32-(8*i))),yy+2,names[zz]);
                  zz++;
               }
            }
            break;
           case 2 :
            if(xx>250) {
               for(i=0;i<2;i++) {
                  charac((xx+4+(8*i)),yy-10,names[zz]);
                  zz++;
               }
            }
            else {
```

```
                    for(i=0;i<2;i++) {
                        charac((xx-(32-(8*i))),yy-10,names[zz]);
                        zz++;
                    }
                }
                break;
        }
return;
}


link() {
     /* function links nodes by information stored in iotbl */
     int a,b,in[40],out[40],plotin[40],plotout[40];
     int content,ktr,i,k,m,maxinctr,maxoutctr,l,p,inx,iny,outx,outy;

     color(14);
     content=iotbl[1];
     ktr=1;
     while(content!=0) {
          maxinctr=iotbl[ktr];
          for(i=0;i<content;i++) {
               ktr++;
               in[i]=iotbl[ktr];
          }
          ktr++;
          content=iotbl[ktr];
          maxoutctr=iotbl[ktr];
          for(i=0;i<content;i++) {
               ktr++;
               out[i]=iotbl[ktr];
          }
          for(p=0;p<maxinctr;p++) {
               plotin[p]=0;
               for(l=0;l<nbrplot;l++) {
                    if(in[p]==linktbl[l][0]) {
                         plotin[p]=1;
                         l=nbrplot;
                    }
               }
          }

          for(p=0;p<maxoutctr;p++) {
               plotout[p]=0;
               for(l=0;l<nbrplot;l++) {
                    if(out[p]==linktbl[l][0]) {
                         plotout[p]=1;
                         l=nbrplot;
                    }
               }
          }

          for(k=0;k<maxinctr;k++) {
               for(i=0;i<maxoutctr;i++) {
                    if(plotin[k]==1 && plotout[i]==1) {
                         a=in[k];
                         b=out[i];
                         for(l=0;l<nbrplot;l++) {
                              if(linktbl[l][0]==a) {
                                   inx=linktbl[l][1];
```

```
                                        iny=linktbl[l][2];
                                        l=nbrplot;
                                   }
                              }
                              for(m=0;m<nbrplot;m++) {
                                   if(linktbl[m][0]==b) {
                                        outx=linktbl[m][1];
                                        outy=linktbl[m][2];
                                        m=nbrplot;
                                   }
                              }
                              lines(inx,iny,outx,outy);
                         }
                    }
          }
     ktr++;
     content=iotbl[ktr];
     }
return;
}


lines(x1,y1,x2,y2) {
     int lnkcase,x,y;

     if(x1<=x2) {
          if(y1<=y2) {
               if(x1==x2) lnkcase=3;
               else {
                         if(y1==y2) lnkcase=0;
                         else lnkcase=4;
               }
          }
          else {
               if(x1==x2) lnkcase=1;
               else lnkcase=5;
          }
     }
     else {
          if(y1<=y2) {
               if(y1==y2) lnkcase=2;
               else lnkcase=7;
          }
          else lnkcase=6;
     }
     switch(lnkcase) {
          case 0:
               segmnt(x1+20,y1,x2-20,y2);
               x=x2-20;
               y=y2;
               segmnt(x-4,y-4,x,y);
               segmnt(x-4,y+4,x,y);
               break;
          case 1:
               segmnt(x1,y1-20,x2,y2+20);
               x=x2;
               y=y2+20;
               segmnt(x-4,y+4,x,y);
               segmnt(x+4,y+4,x,y);
               break;
```

```
        case 2:
             segmnt(x1-20,y1,x2+20,y2);
             x=x2+20;
             y=y2;
             segmnt(x+4,y-4,x,y);
             segmnt(x+4,y+4,x,y);
             break;
        case 3:
             segmnt(x1,y1+20,x2,y2-20);
             x=x2;
             y=y2-20;
             segmnt(x-4,y-4,x,y);
             segmnt(x+4,y-4,x,y);
             break;
        case 4:
             segmnt(x1+24,y1+24,x2-24,y2-24);
             x=x2-16;
             y=y2-16;
             segmnt(x,y-6,x,y);
             segmnt(x-6,y,x,y);
             segmnt(x2-24,y2-24,x,y);
             break;
        case 5:
             segmnt(x1+24,y1-24,x2-24,y2+24);
             x=x2-16;
             y=y2+16;
             segmnt(x,y+6,x,y);
             segmnt(x-6,y,x,y);
             segmnt(x2-24,y2+24,x,y);
             break;
        case 6:
             segmnt(x1-24,y1-24,x2+24,y2+24);
             x=x2+16;
             y=y2+16;
             segmnt(x,y+6,x,y);
             segmnt(x+6,y,x,y);
             segmnt(x2+24,y2+24,x,y);
             break;
        case 7:
             segmnt(x1-24,y1+24,x2+24,y2-24);
             x=x2+16;
             y=y2-16;
             segmnt(x,y-6,x,y);
             segmnt(x+6,y,x,y);
             segmnt(x2+24,y2-24,x,y);
             break;
    }
 return;
 }


 select()  {
     int i,n;
     char v;

     space(2);
     printf("***->THERE ARE 3 VERSIONS TO THIS GRAPHICS PACKAGE \n");
     printf("     PLEASE SELECT ONE OF THE FOLLOWING VERSIONS: \n");
   again:space(2);
     printf("      VERSION 1 ... PETRI-NET PACKAGE ....... TYPE 1 \n");
```

```
    printf("       VERSION 2 ... PACKET REPRESENTATION ... TYPE 2 \n");
    printf("       VERSION 3 ... MULTIROUTING PACKAGE .... TYPE 3 \n");
    n = 0;
  twice:space(2);
    printf("***->");
    n++;
    i = 0;
    while ((v=getchar()) != '\n') {
        vbuf[i] = v;
        i++;
    }
    vbuf[i] = '\0';

    i = 0;
    while(vbuf[i] != '\0')  {
        v = vbuf[i];
        if(n==1) {
              switch(v) {
                case '1':
                 vers = 1;
                 break;
                case '2':
                 vers = 2;
                 break;
                case '3':
                 vers = 3;
                 break;
                default:
                 printf
                 ("***->incorrect version try again!");
                 goto again;
                 break;
              }
              i++;
        }
        else {
              switch(v) {
                case '0':
                 set = 0;
                 break;
                case '1':
                 set = 1;
                 break;
                case '2':
                 set = 2;
                 break;
                default:
                 printf
                 ("***->incorrect genisco set try again");
                 printf
                 ("      set selection should be 0,1,or2");
                 n = 1;
                 goto twice;
                 break;
              }
              i++;
        }
    }
    if(n==1) {
         space(2);
```

```
            printf("***->NOW SELECT THE GENISCO SET YOU WISH \n");
            printf("     THE PROGRAM TO BE DISPLAYED TO..... \n");
            printf("     IN C3 LAB EITHER SET0, SET1 OR SET2 \n");
            goto twice;
     }
return;
}


imark() {
     /* marks initial state of system by calling appropriate function */
     int p,colour,e,g,x,y;

     bp1 = file1;
     dfltcolor=3;
     color(dfltcolor);
     colour=2;
     ctroverflow=0;
     while ((g=(bp1->nameptr)) !=0) {
          if((p=(bp1->plot)) !=0) {
                 switch(vers)  {
                        case 1:
                            ivers1();
                            break;
                        case 2:
                            ivers2(colour);
                            break;
                        case 3:
                            ivers3(colour);
                            break;
                 }
          }
          if(dfltcolor!=3) color(3);
          bp1++;
     }
     color(14);
     printg(0,350.,480.,"TIME FRAME = 1");
     preread(1);
     hilite ();
     displa();
     hold();
     reset(1);
     color(13);
     ovrflow();
     printg(0,350.,480.,"TIME FRAME = 1");
return;
}


hold() {
     int i;

    time:space(2);
     printf("***->THIS IS THE INITIAL STATE OF THE NETWORK \n");
     printf("     ENTER THE TIMING MODE FOR EXECUTION .... \n");
     printf("     '0' FOR NO DELAYS ..'1' FOR FRAME PAUSES \n");
     printf("***->");
     i = 0;
     while((timing=getchar()) != '\n') {
          tbuf[i] = timing;
```

```
            i++;
        }
        tbuf[i] = '\0';

        i = 0;
        while(tbuf[i] != '\0')  {
            timing = tbuf[i];
            i++;
        }
        if(timing!='0' && timing!='1') {
            printf("***->incorrect version try again!");
            goto time;
        }
    return;
    }


    ivers1()  {
        int e,x,y,z;
        char check;

        e=(bpl->marker);
        x=(bpl->xcord);
        y=(bpl->ycord);
        z=(bpl->nameptr);
        if((check=names[z+1])!='I' && (check=names[z+1])!='0') {
            printg(0,x-3.0,511.-(y-3),"%d",e);
        }
        else {
            if(check=names[z+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",e);
            else printg(0,x-14.0,511.-(y-9),"%d",e);
        }
    return;
    }


    ivers2(colour)  {
        int e,x,y,z;
        char check;

        e=(bpl->marker);
        x=(bpl->xcord);
        y=(bpl->ycord);
        z=(bpl->nameptr);
        if((check=names[z+1])!='I' && (check=names[z+1])!='0') {
            pckt2(x,y,e,colour);
        }
        else {
            if(check=names[z+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",e);
            else printg(0,x-14.0,511.-(y-9),"%d",e);
        }
    return;
    }


    ivers3(colour)  {
        char  keep,*kptr;
        int a,aa,b,bb,c,cc,clr[fired],i,k,n,stack,total,x,y;

        total=0;
```

```
     n=1;
     x=(bp1->xcord);
     y=(bp1->ycord);
     a=(bp1->length);
     b=(bp1->nameptr);
     c=(bp1->marker);
     total=total+c;

     if(c>0)  {
          for(i=0;i<c;i++)  {
               kptr = &names[b+a-2];
               keep = *kptr;
               clr[n]=atoi(&keep);
               n++;
          }
     }

     kptr = &names[b+a];
     keep = *kptr;
     stack=atoi(&keep);

     if(names[b+(a-1)]!='0') {
          if(names[b+(a-1)]=='2')  stack=stack+20;
          else  stack = stack + 10;
     }

     for(i=0;i<stack-1;i++)  {
          bp1++;
          aa=(bp1->length);
          bb=(bp1->nameptr);
          cc=(bp1->marker);
          total=total+cc;
          if(cc>0)  {
               for(k=0;k<cc;k++)  {
                    kptr = &names[bb+aa-2];
                    keep = *kptr;
                    clr[n]=atoi(&keep);
                    n++;
               }
          }
      }
      if(names[b+1]!='I' && names[b+1]!='0') {
        pckt3
        (x,y,total,clr[1],clr[2],clr[3],clr[4],clr[5],clr[6],clr[7],
         colour);
      }
      else {
           if(names[b+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",total);
           else printg(0,x-14.0,511.-(y-9),"%d",total);
      }
return;
}


preread(flag)  {
       int bucket[2],count,fd,fa,i,nbrtrns;
       if(flag!=3) {
      if(flag==1)  {
           fd = open (fbuf, 0);
           if (fd<=0) {
```

```
                printf("***->error occurred in opening fd file");
            }
            fdfbuf=fd;
            fg = open (gbuf, 0);
            if (fg<=0) {
                printf("***->error occurred in opening fg file");
            }
            fdgbuf=fg;
        }
        if((count=read (fdfbuf, bucket, 2))!=2) {
            printf("***->error occurred in fd bucket read");
        }
        if((count=read (fdfbuf, buffer,(ievents*2)))!=(ievents*2)) {
            printf("***->error occurred in buffer read");
        }
        if((count=read (fdgbuf, bucket, 2))!=2) {
            printf("***->error occurred in fg bucket read");
        }
        if((count=read (fdgbuf, cntrl,2))!=2) {
            printf("***->error occurred in cntrl read");
        }
        if(cntrl[0]==0) {
            space(2);
            printf("***->the last network state has been achieved");
            kthframe=frames+1;
            space(2);
        }
        else {
            if((count=read (fdgbuf, bucket, 2))!=2) {
                  printf("***->error occurred in bucket read");
            }
            nbrtrns = cntrl[0]*2;
            if((count=read (fdgbuf, firing,nbrtrns))!= nbrtrns) {
                  printf("***->error occurred in firing read");
            }
        }
         }
         else {
        close(fdfbuf);
        close(fdgbuf);
         }
return;
}


hilite() {
        int q,i,in[25],intbl,j,k,l,m,maxinctr,maxoutctr,n,out[25],
            outtbl,p,plotin[25],plotout[25],inx,iny,outx,outy;

        tapcount=0;
        for(i=0;i<cntrl[0];i++) {
            bp2=file2;
            for(j=0;j<firing[i]-1;j++) {
                bp2++;
            }
            intbl=(bp2->intrn);
            outtbl=(bp2->outtrn);
            maxinctr=iotbl[intbl];
            for(k=0;k<maxinctr;k++) {
                intbl++;
```

```
                in[k]=iotbl[intbl];
            }
            maxoutctr=iotbl[outtbl];
            for(l=0;l<maxoutctr;l++) {
                outtbl++;
                out[l]=iotbl[outtbl];
            }
            for(p=0;p<maxinctr;p++) {
                plotin[p]=0;
                for(l=0;l<nbrplot;l++) {
                    if(in[p]==linktbl[l][0]) {
                        plotin[p]=1;
                        l=nbrplot;
                    }
                }
            }

            for(p=0;p<maxoutctr;p++) {
                plotout[p]=0;
                for(l=0;l<nbrplot;l++) {
                    if(out[p]==linktbl[l][0]) {
                        plotout[p]=1;
                        l=nbrplot;
                    }
                }
            }
            for(k=0;k<maxinctr;k++) {
                for(g=0;g<maxoutctr;g++) {
                    if(plotin[k]==1 && plotout[g]==1) {
                        for(m=0;m<nbrplot;m++) {
                            if(linktbl[m][0]==in[k]) {
                                inx=linktbl[m][1];
                                iny=linktbl[m][2];
                                m=nbrplot;
                            }
                        }
                        for(n=0;n<nbrplot;n++) {
                            if(linktbl[n][0]==out[g]){
                                outx=linktbl[n][1];
                                outy=linktbl[n][2];
                                n=nbrplot;
                            }
                        }
                        color(11);
                        lines(inx,iny,outx,outy);
                        xinstore[tabcount]=inx;
                        yinstore[tabcount]=iny;
                        xoutstore[tabcount]=outx;
                        youtstore[tabcount]=outy;
                        tabcount++;
                    }
                }
            }
      }
return;
}


lnkrvt() {
      int i,inx,iny,outx,outy;
```

```

     color(14);
     for(i=0;i<tabcount;i++) {
          inx= xinstore[i];
          iny= yinstore[i];
          outx= xoutstore[i];
          outy= youtstore[i];
          lines(inx,iny,outx,outy);
     }
return;
}


stage() {

     if (kthframe!=0 && kthframe<frames+1) {
          preread(2);
          if(kthframe!=frames+1) {
               hilite();
               if(timing=='1') {
                    displa();
                    pause(1);
               }
          }
     }
return;
}


marking() {
     /* function displays successive iterations of the network */
     int colour,draw,i,mark,n,x,y;

     bpl = file1;
     n = 2;

     /* following loop processes ievent # data segments each pass */
     for(kthframe=0;kthframe<frames;kthframe++) {
          stage();
          if(kthframe>0) {
               reset();
               color(13);
               ovrflow();
               printg(0,350.,490.,"TIME FRAME = %d",n);
               n++;
          }
          iflag = kthframe;
          draw = (rel->plot);
          dfltcolor=3;
          color (dfltcolor);
          colour=2;
          ctroverflow=0;
          for (i=0; i < ievents; i++)  {
               if (draw == 1) {
                    ictr = i;
                    switch (vers)  {
                         case 1:
                              vers1();
                              break;
                         case 2:
```

```
                            vers2(colour);
                            break;
                        case 3:
                            vers3(colour);
                            break;
                    }
                }
                i = ictr++;
                if(dfltcolor!=3) color(3);
                bpl++;
                draw = (bpl -> plot);
            }
            dfltcolor=14;
            color(dfltcolor);
            colour=13;
            printg(0,350.,480.,"TIME FRAME = %d",n);
            if(timing=='1') pause(2);
            lnkrvt();
            bpl = file1;
        }
        preread(3);
return;
}


ovrflow() {

        int i,x,y;

        i=0;
        while(overflowtbl[i][0]!=0) {
            x=overflowtbl[i][0];
            y=overflowtbl[i][1];
            if(x>250) {
            block((x+38)*1.,511.-(y+10)*1.,(x+54)*1.,511.-y*1.);
            }
            else {
            block((x-50)*1.,511.-(y+10)*1.,(x-35)*1.,511.-y*1.);
            }
            overflowtbl[i][0]=0;
            overflowtbl[i][1]=0;
            i++;
        }
return;
}


vers1()  {
        int e,x,y,z;
        char check;

        e=buffer[ictr];
        x=(bpl->xcord);
        y=(bpl->ycord);
        z=(bpl->nameptr);
        if((check=names[z+1])!='I' && (check=names[z+1])!='O') {
             printg(0,x-3.0,511.-(y-3),"%d",e);
        }
        else {
             if(check=names[z+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",e);
```

```
                else printg(0,x-14.0,511.-(y-9),"%d",e);
        }
return;
}


vers2(colour)   {
        int mark,x,y,z;
        char check;

        x=(bp1->xcord);
        y=(bp1->ycord);
        z=(bp1->nameptr);
        mark=buffer[ictr];
        if((check=names[z+1])!='I' && (check=names[z+1])!='0') {
         pckt2(x,y,mark,colour);
        }
        else {
         if((check=names[z+1])=='I') printg(0,x-14.0,511.-(y+2),"%d",mark);
         else printg(0,x-14.0,511.-(y-9),"%d",mark);
        }
return;
}


vers3(colour)   {
        int a,aa,b,bb,mark,marks,clr[fired],j,k,n,stack,total,x,y;
        char keep,*kptr;

        total=0;
        n=1;
        x=(bp1->xcord);
        y=(bp1->ycord);
        a=(bp1->length);
        b=(bp1->nameptr);
        mark=buffer[ictr];
        total=total+mark;
        if(mark>0)   {
            for(j=0;j<mark;j++)   {
                kptr = &names[b+a-2];
                keep = *kptr;
                clr[n]=atoi(&keep);
                n++;
            }
        }

        kptr = &names[b+a];
        keep = *kptr;
        stack=atoi(&keep);

        if(names[b+(a-1)]!='0') {
            if(names[b+(a-1)]=='1')  stack=stack+10;
            else {
                  if(names[b+(a-1)]=='2')  stack=stack+20;
                  else  stack = stack + 30;
            }
        }
        for(j=0;j<stack-1;j++)   {
            bp1++;
            ictr++;
```

```
            aa=(bpl->length);
            bb=(bpl->nameptr);
            marks=buffer[ictr];
            total=total+marks;
            if(marks>0)  {
                for(k=0;k<marks;k++)  {
                    kptr = &names[bb+aa-2];
                    keep = *kptr;
                    clr[n]=atoi(&keep);
                    n++;
                }
            }
        }
        if(names[b+1]!='I' && names[b+1]!='0') {
          pckt3
          (x,y,total,clr[1],clr[2],clr[3],clr[4],clr[5],clr[6],clr[7],
          colour);
        }
        else {
          color(3);
          if(names[b+1]=='I') printg(0,x-14.0,511.-(y+2),"%d",total);
          else printg(0,x-14.0,511.-(y-9),"%d",total);
        }
 return;
 }


 pckt2(xaxis,yaxis,point,class) {

  switch(point) {
    case 0:
     break;
    case 1:
     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
     break;
    case 2:
     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
     break;
    case 3:
     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
     break;
    case 4:
     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
     break;
    case 5:
     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
     block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
     block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
     block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
     break;
    case 6:
     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
```

```
      block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
      block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
      block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
      block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
      break;
    case 7:
      block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
      block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
      block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
      block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
      block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
      block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
      block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
      break;
    default:
      overflowtbl[ctroverflow][0]=xaxis;
      overflowtbl[ctroverflow][1]=yaxis;
      ctroverflow++;
      block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
      block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
      block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
      block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
      block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
      block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
      block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
      color(class);
      dfltcolor=class;
      if(xaxis>250) {
           printg(0,xaxis+38.0,511.-yaxis,"%d",point-7);
      }
      else {
           printg(0,xaxis-50.0,511.-yaxis,"%d",point-7);
      }
      break;
  }
return;
}



ockt3(xaxis,yaxis,total,c1,c2,c3,c4,c5,c6,c7,class)  {

 switch (total)  {
   case 0:
     break;
   case 1:
     color(c1);
     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
     break;
   case 2:
     color(c1);
     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
     color(c2);
     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
     break;
   case 3:
     color(c1);
     block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
     color(c2);
     block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
```

```
        color(c3);
        block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
        break;
     case 4:
        color(c1);
        block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
        color(c2);
        block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
        color(c3);
        block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
        color(c4);
        block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
        break;
     case 5:
        color(c1);
        block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
        color(c2);
        block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
        color(c3);
        block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
        color(c4);
        block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
        color(c5);
        block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
        break;
     case 6:
        color(c1);
        block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
        color(c2);
        block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
        color(c3);
        block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
        color(c4);
        block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
        color(c5);
        block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
        color(c6);
        block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
        break;
     case 7:
        color(c1);
        block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
        color(c2);
        block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
        color(c3);
        block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
        color(c4);
        block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
        color(c5);
        block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
        color(c6);
        block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
        color(c7);
        block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
        break;
     default:
        overflowtbl[ctroverflow][0]=xaxis;
        overflowtbl[ctroverflow][1]=yaxis;
        ctroverflow++;
        color(c1);
```

```
	block((xaxis-3)*1.,511.-(yaxis+3)*1.,(xaxis+3)*1.,511.-(yaxis-3)*1.);
	color(c2);
	block((xaxis-3)*1.,511.-(yaxis-7)*1.,(xaxis+3)*1.,511.-(yaxis-13)*1.);
	color(c3);
	block((xaxis-3)*1.,511.-(yaxis+13)*1.,(xaxis+3)*1.,511.-(yaxis+7)*1.);
	color(c4);
	block((xaxis+7)*1.,511.-(yaxis+3)*1.,(xaxis+13)*1.,511.-(yaxis-3)*1.);
	color(c5);
	block((xaxis-13)*1.,511.-(yaxis+3)*1.,(xaxis-7)*1.,511.-(yaxis-3)*1.);
	color(c6);
	block((xaxis-13)*1.,511.-(yaxis+13)*1.,(xaxis-7)*1.,511.-(yaxis+7)*1.);
	color(c7);
	block((xaxis+7)*1.,511.-(yaxis+13)*1.,(xaxis+13)*1.,511.-(yaxis+7)*1.);
	color(class);
	afltcolor=class;
	if(xaxis>250) {
		printg(0,xaxis+38.0,511.-yaxis,"%d",total-7);
	}
	else {
		printg(0,xaxis-50.0,511.-yaxis,"%d",total-7);
	}
	break;
    }
return;
}


reset() {
	/* reset function for successive network iterations */
	int i,mark,x,y,z;

	if(vers==1 || vers==2) {
	    for(i=0;i<nbrplot;i++)  {
	    color(14);
	    x=linktbl[i][1];
	    y=linktbl[i][2];
	    z=linktbl[i][3];
	    if(z==0) {
		block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
		block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
	    }
	    else {
		block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
		block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
	    }
	    }
	}
	else {
	    for(i=0;i<tblctr-1;i++) {
	    color(14);
	    x=uniquetbl[i][1];
	    y=uniquetbl[i][2];
	    z=uniquetbl[i][3];
	    if(z==0) {
		block((x-3)*1.,511.-(y-7)*1.,(x+3)*1.,511.-(y-13)*1.);
		block((x-13)*1.,511.-(y+13)*1.,(x+13)*1.,511.-(y-3)*1.);
	    }
	    else {
		block((x-16)*1.,511.-(y-2)*1.,x*1.,511.-(y-10)*1.);
		block((x-16)*1.,511.-(y+10)*1.,x*1.,511.-(y+2)*1.);
```

```
            }
            }
        }
 return;
 }


 coltab()  {
        int i;

        i=11*16;
        lodcol(i++,15,15,15);   /*  color 0  */
        lodcol(i++,0,10,0);     /*  color 1  */
        lodcol(i++,15,0,0);     /*  color 2  */
        lodcol(i++,15,15,0);    /*  color 3  */
        lodcol(i++,12,0,12);    /*  color 4  */
        lodcol(i++,5,5,12);     /*  color 5  */
        lodcol(i++,6,6,5);      /*  color 6  */
        lodcol(i++,5,3,3);      /*  color 7  */
        lodcol(i++,10,6,10);    /*  color 8  */
        lodcol(i++,12,5,5);     /*  color 9  */
        lodcol(i++,5,5,3);      /*  color 10 */
        lodcol(i++,12,12,0);    /*  color 11 */
        lodcol(i++,8,7,3);      /*  color 12 */
        lodcol(i++,5,4,2);      /*  color 13 */
        lodcol(i++,0,0,6);      /*  color 14 */
        lodcol(i++,6,0,0);      /*  color 15 */
 return;
 }


            /******************************************************/
            /********                                  **********/
            /********     END OF PROGRAM LINKGRAPH.C   **********/
            /********                                  **********/
            /******************************************************/

```

LIST OF REFERENCES

1. Abramson, Norman and Franklin F. Kuo, Computer-Communication Networks, Prentice-Hall, 1973.

2. Anderson, George A, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", Computing Surveys, Vol. 7, No. 4, Dec 1975.

3. Awad, Elias M., Systems Analysis and Design, Richard D. Irwin, Inc., 1979.

4. Browne, J. C., "Performance Analysis and Evaluation: The Connection to Reality", Research Directions in Software Technology, The MIT PRESS, 1979.

5. Cox, L. A. Jr., "Performance Prediction of Computer Architectures Operating on Linear Mathematical Models", Ph.D. Thesis, Computer Science Dept., UC Davis Report UCRL-52582 (Sept 28,1978).

6. Cypser, R. J., Communications Architectures for Distributed Systems, Addison-Wesley Publishing Company, 1978.

7. Davies, D. W., and others, Computer Networks and Their Protocol, John Wiley and Sons, 1979.

8. Dixon, F., "The Impact of Technology on the Future Tactical Air Control Systems", Air Command and Staff College, March 1977.

9. Enslow, Philip H. Jr., "Multiprocessor Organization-A Survey", Computing Surveys, Vol. 9, No. 1, March, 1977.

10. Hamming, R. W., "Why Do You Believe the Simulation is Relevant?", Simulation, Naval Post Graduate School, 1979.

11. Heitmeyer, Constance L., and others, "A Survey of Packet Switching Techniques for Broadcast Media", NRL Report 8035, Naval Research Laboratory Washington D.C., Oct 12, 1776.

12. Hopper, Grace, M., "David and Goliath", Selected Computer Articles 78, Dept. of Defense Computer Institute, 1978.

13. Kahn, Robert E., "The Organization of Computer Resources into a Packet Radio Network", IEEE Transactions on Communications, Vol. Com-25, No. 1, Jan 1977.

14. Kleinrock, Leonard, "Performance of Distributed Multi-Access Computer-Communication Systems", Proceedings of IFIP Congress 1977, Toronto, Canada, Aug 1977.

15. Kleinrock, Leonard, "Principles and Lessons in Packet Communications", Proceedings of the IEEE, Vol. 66, No. 11, Nov 1978.

16. Kunzelman, Ronald C., and others, "Packet Radio Experimental Network Research and Development, Final Technical Report", SRI International, Jan 1978.

17. Lawson, Billy R., "A Testbed for Evaluation of Packet Radio in an Army Tactical Corps", U.S.Army letter.

18. Moli, Gesualdo De, "On Networking", Computer Networks and Simulation, North-Holland, 1978.

19. Peterson, James L., "Petri-Nets", Computing Surveys, Vol. 9, No. 3, Sept 1977.

20. Price, W. L., Computer Networks and Simulation, North-Holland Publishing Company, 1978.

21. Stowers, Douglas M., "Computer Architecture Performance Prediction for Naval Fire Control Systems", Naval Post Graduate School Thesis, Dec 1979.

22. Terman, Lewis, M., "The Role of Microelectronics in Data Processing", Scientific American, Sept 1977.

23. Thurber, Kenneth J., and Leon D. Wald, "Associative and Parallel Processors", Computing Surveys, Vol. 7, No. 4, Dec 1975.

24. FM-24-1, Combat Communications, U.S. Army Field Manual.

25. Letter of Agreement (LOA) for PLRS/JTIDS Hybrid, U.S. Army Training and Doctrine Command, ACN 58401, Dept of the Army, 6 July 1979.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	I. Larry Avrunin DTNSRDC Code 18 Bethesda, Maryland 20084	1
2.	Mr. R. P. Crabb Code 9134 Naval Ocean Systems Center San Diego, California 92152	1
3.	Mr. C. H. Gleissner DTNSRDC Code 18 Bethesda, Maryland 20084	1
4.	Ms. Kathryn Heninger Code 7503 Naval Research Laboratory Washington, D. C. 20375	1
5.	Mr. Roanld P. Kasik Code 4451 Naval Underwater Systems Center Newport, Rhode Island 02840	1
6.	Dr. J. McGraw U. C. - L.L.L. (L-794) P.O. Box 808 Livermore, California 94550	1
7.	Mr. Henry G. Stuebing Code 503 Naval Air Development Center Warminster, Pennsylvannia 18974	1
8.	Mr. Joel Trimble Code 221 Office of Naval Research 800 North Quincy Arlington, Virginia 22217	1
9.	Mr. Mark Underwood Code P204 NPROC San Diego, California 92152	1

10. Mr. Michael Wallace 1
DTNSRDC
Code 1828
Bethesda, Maryland 20084

11. Mr. Walter P. Warner 1
NSWC
Code K70
Dahlgren, Virginia 22448

12. Dr. John Zenor 1
Code 31302
Naval Weapons Center
China Lake, California 93555

13. Office of Research Administration 1
Code 012A
Naval Postgraduate School
Monterey, California 93940

14. Defense Documentation Center 2
Cameron Station
Alexandria, Virginia 22314

15. Library, Code 0142 2
Naval Postgraduate School
Monterey, California 93940

16. Department Chairman, Code 52 2
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940

17. Prof. L. Cox, Code 52Cl 5
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940

18. Prof. J. Wozencraft, Code 74 2
Chairman, C3 Academic Group
Naval Postgraduate School
Monterey, California 93940

19. Prof. R. Schell, Code 52Sj 1
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940

20. Prof. O. Kodres, Code 52Kr 1
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940

21. Cpt. Robert J. Hartel 5
Code 110
Defense Communications Agency
Washington D.C. 20305

22. Capt. Stephen C. Jennings 2
MCDEC
Communication Officers School
Quantico, Virginia 22134

23. Mr. Douglas M. Stowers 1
Naval Sea Systems Command
Code Sea 62Y21D
Washington D.C. 20362

Thesis 189382
J4726 Jennings
c.1 Petri-Net simulations of communications networks.